Università
della
Svizzera
italiana

**Faculty
of Informatics**

EuroCG 2016

32nd European Workshop on Computational Geometry March 30 – April 1, 2016

Book of Abstracts

http://www.eurocg2016.usi.ch/

Preface

The 32nd European Workshop on Computational Geometry (EuroCG '16), was held at the Università della Svizzera italiana (USI), Lugano, Switzerland, on March 30 – April 1, 2016.

EuroCG is an annual workshop that combines a strong scientific tradition with a friendly and informal atmosphere. The workshop is a forum where researchers can meet, discuss their work, present their results, and establish scientific collaborations, in order to promote research in the field of Computational Geometry, within Europe and beyond.

We received 77 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 65 submissions for presentation at the workshop. EuroCG does not have formally published proceedings; therefore, we expect most of the results presented here to be also submitted to peer-reviewed conferences and/or journals. This book of abstracts, available through the EuroCG '16 web site http://www.eurocg2016.usi.ch, should be regarded as a collection of preprints. In addition to the 65 contributed talks, this book also contains abstracts of the three invited lectures, given by Dan Halperin, Dorothea Wagner, and Emo Welzl.

Many thanks to all authors, speakers, and invited speakers for their participation, and to the members of the program committee and all external reviewers for their insightful comments. We gratefully thank the sponsors of EuroCG '16 for their support which made this event possible and helped keep the registration fees low: the Università della Svizzera italiana, the Hasler Foundation, and the Swiss National Science Foundation for sponsoring the travel of the invited speakers. Special thanks to all members of the organizing committee and members of the Dean's office at the Università della Svizzera italiana, for their work that made EuroCG '16 possible.

March 2016

Evanthia Papadopoulou
EuroCG '16 Chair

Organizing Committee

- Gill Barequet, Technion, Israel
- Elena Khramtcova, Università della Svizzera italiana (USI), Switzerland
- Matias Korman, Tohoku University, Japan
- Evanthia Papadopoulou, Università della Svizzera italiana (USI), Switzerland
- Rodrigo I. Silveira, Universitat Politècnica de Catalunya (UPC), Spain

Program Committee

- Gill Barequet (co-chair), Technion, Israel
- Mark de Berg, TU Eindhoven, The Netherlands
- Sergio Cabello, University of Ljubljana, Slovenia
- Ioannis Emiris, University of Athens, Greece
- Dan Halperin, Tel-Aviv University, Israel
- Martin Held, University of Salzburg, Austria

- Michael Hoffmann, ETH Zurich, Switzerland
- Matya Katz, Ben-Gurion University, Israel
- Matias Korman, Tohoku University, Japan
- Marc van Kreveld, Utrecht University, The Netherlands
- Giuseppe Liotta, University of Perugia, Italy
- Wolfgang Mulzer, Freie Universität Berlin, Germany
- Bengt Nilsson, Malmö University, Sweden
- Evanthia Papadopoulou (co-chair), Università della Svizzera italiana (USI), Switzerland
- Günter Rote, Freie Universität Berlin, Germany
- Vera Sacristán, Universitat Politècnica de Catalunya (UPC), Spain
- Maria Saumell, University of West Bohemia, Czech Republic
- Rodrigo I. Silveira, Universitat Politècnica de Catalunya (UPC), Spain
- Bettina Speckmann, TU Eindhoven, The Netherlands
- Monique Teillaud, INRIA Nancy - Grand Est, France
- Birgit Vogtenhuber, TU Graz, Austria

External Reviewers

- Oswin Aichholzer
- Franz Aurenhammer
- Aritra Banik
- Bahareh Banyassady
- Olivier Devillers
- Yago Diez Donoso
- Guenther Eder
- Jiri Fiala
- Vissarion Fisikopoulos
- Elena Khramtcova
- Sylvain Lazard
- Guillaume Moroz
- Peter Palfrader
- Irene Parada
- Alexander Pilz
- Simon Pratt
- Ioannis Psarros
- Patrick Schnider
- Paul Seiferth
- Kiril Solovey
- Yannik Stein
- Hans Raj Tiwary
- Kevin Verbeek
- Manuel Wettstein

Table of Contents

Session 4A

Session 4B

Session 5A

Session 7A

Session 7B

Hard versus Easy in Robot Motion Planning: Closing the Ring

Dan Halperin*

Abstract

Early results in robot motion planning had forecast a bleak future for the field by showing that problems with many degrees of freedom are intractable. Then came sampling-based planners that have been successfully, and often easily, solving a large variety of problems with many degrees of freedom.

We strive to formally determine what makes a motion-planning problem with many degrees of freedom easy or hard. I'll describe our quest to resolve this (still wide open) problem, and some progress we have made in the context of multi-robot motion planning.

*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: danha@post.tau.ac.il

Route Planning in Transportation—When the Metric Matters

Dorothea Wagner*

Abstract

Nowadays, route planning systems belong to the most frequently used information systems at all. The algorithmic core problem of such systems is the classical shortest paths problem that can be solved by Disjkstra's algorithm in almost linear time. However, Dijkstra's algorithm still takes a few seconds in continental-sized graphs, which is too slow for practical scenarios. Algorithms for route planning in transportation networks have recently undergone a rapid development, leading to methods that are up to several million times faster than Dijkstra's algorithm. The metric considered is typically driving time, but could be also something else like distance, costs, number of left turns or energy consumption. This talk provides a condensed survey of recent advances in algorithms for route planning in transportation networks. In particular, we will discuss scenarios where the metric matters, including customizable route planning which supports, e.g., real-time traffic updates.

*Institut für Theoretische Informatik, Karlsruher Institut für Technologie, D-76128 Karlsruhe, Germany. E-mail: dorothea.wagner@kit.edu

From Crossing-Free Graphs on Wheel Sets to Polytopes with Few Vertices

Emo Welzl*

Abstract

A perfect matching on a finite planar point set S is crossing-free if all of its edges are disjoint in the straight-line embedding on S. In 1948 Motzkin was interested in the number of such crossing-free perfect matchings for S the $2m$ vertices of a convex polygon and he proved that to be the m-th Catalan number.

S is called a wheel set if all but exactly one point in S are vertices of its convex hull. Again we start by asking for the number of crossing-free perfect matchings of such a wheel set S, going the smallest possible step beyond Motzkins endeavor. Since position matters now, in the sense that the number is not determined by the cardinality of the wheel set alone, this immediately raises extremal and algorithmic questions. Answering these comes with all kinds of surprises. In fact, it turns out that for the purpose of counting crossing-free geometric graphs (of any type, e.g. triangulations or crossing-free spanning trees) on such a set P it suffices to know the so-called frequency vector of P (as opposed to the full order type information) – a simple formula dependent on this frequency vector exists. Interestingly, the number of order types of n points in almost convex position is roughly 2^n, compared to the number of frequency vectors which is about $2^{n/2}$.

Finally, this takes us on a journey to the rectilinear crossing-number of the complete graph, to counting of origin-embracing triangles and simplices (simplicial depth) and to counting facets of high-dimensional polytopes with few vertices.

(Based on recent joint work with Andres J. Ruiz-Vargas, Alexander Pilz, and Manuel Wettstein.)

*Department of Computer Science, Institute of Theoretical Computer Science, ETH Zürich, Switzerland. E-mail: emo@inf.ethz.ch

Grouping Time-varying Data for Interactive Exploration

Arthur van Goethem* Marc van Kreveld† Maarten Löffler† Bettina Speckmann* Frank Staals‡

1 Introduction

We present algorithms and data structures that support the interactive analysis of the grouping structure of one-, two-, or higher-dimensional time-varying data while varying all defining parameters. Grouping structures (which track the formation and dissolution of groups) characterize important patterns in the evolution of sets of time-varying data. We follow Buchin et al. [4] who define groups using three parameters: group-size, group-duration, and inter-entity distance.

Trajectory grouping structure [4]. Let $\mathcal{X}$ be a set of n entities moving in $\mathbb{R}^d$ and let $\mathbb{T}$ denote time. The entities trace trajectories in $\mathbb{T} \times \mathbb{R}^d$. We assume that each individual trajectory is piecewise linear and consists of at most τ vertices. Two entities a and b are *ε-connected* at time t if there is a chain of entities $a = c_1, .., c_k = b$ such that for any pair of consecutive entities c_i and c_{i+1} the distance at time t is at most ε. A set G is ε-connected, if for any pair $a, b \in G$, the entities are ε-connected. Given parameters m, ε, and δ, a set of entities G is an (m, ε, δ)*-group* during time interval I if (and only if) (i) G has size at least m, (ii) $\mathit{duration}(I) \geq \delta$, and (iii) G is *ε-connected* at any time $t \in I$. An (m, ε, δ)-group (G, I) is *maximal* if G is maximal in size or I is maximal in duration, that is, if there is no group $H \supset G$ that is also ε-connected during I, and no interval $J \supset I$ such that G is ε-connected during J.

Results and Organisation. We describe a data structure $\mathcal{D}$ that represents the grouping structure, that is, its maximal groups, while allowing efficient change of the parameters. The complexity of the problem appears already in one-dimensional time-varying data. Hence we restrict our description to $\mathbb{R}^1$, the full paper extends our results to higher dimensions.

If all three parameters m, ε, and δ can vary independently the question arises what constitutes a meaningful maximal group. Consider a maximal (m, ε, δ)-group (G, I). If we slightly increase ε to ε', and consider a slightly longer time interval $I' \supseteq I$ then (G, I') is a maximal $(m, \varepsilon', \delta)$-group. Intuitively, these groups (G, I) and (G, I') are the same. Thus, we are interested only in (maximal) groups that are "combinatorially different". The set of entities G may also be a maximal (m, ε, δ)-group during a time interval J completely disjoint from I, we also wish to consider (G, I) and (G, J) to be combinatorially different groups. In Section 2 we formally define when two (maximal) (m, ε, δ)-groups are (combinatorially) different. We prove that there are at most $O(|\mathcal{A}|n^2)$ such groups, where $\mathcal{A}$ is the arrangement of the trajectories in $\mathbb{T} \times \mathbb{R}^1$, and $|\mathcal{A}|$ is its complexity. We also argue that the number of maximal groups may be as large as $\Omega(\tau n^3)$, even for fixed parameters m, ε, and δ and in $\mathbb{R}^1$. This significantly strengthens the lower bound of Buchin et al. [4]. In Section 3 we present an $O(|\mathcal{A}|n^2 \log^2 n)$ time algorithm to compute all combinatorially different maximal groups.

In the full paper we describe a data structure that allows us to efficiently obtain all groups for a given set of parameter values. We also describe data structures for the interactive exploration of the data. Specifically, given the set of maximal (m, ε, δ)-groups we want to change one or more of the parameters and efficiently report only those maximal groups which either ceased to be a maximal group or became one. Our data structures can answer *symmetric-difference queries* [5].

2 Combinatorially Different Maximal Groups

We consider entities moving in $\mathbb{R}^1$, hence the trajectories form an arrangement $\mathcal{A}$ in $\mathbb{T} \times \mathbb{R}^1$. Consider the four-dimensional *parameter space* $\mathbb{P}$ with axes time, size, distance, and duration. A set of entities G defines a region A_G in which it is *alive*: a point $(t, m, \varepsilon, \delta)$ lies in A_G if and only if G is an (m, ε, δ)-group at time t. These regions help define when groups are combinatorially different. We start by fixing $m = 1$ and $\delta = 0$ to define and count the number of combinatorially different maximal $(1, \varepsilon, 0)$-groups, over all choices of parameter ε. Theorem 6 and Lemma 7 extend these results to include other values of δ and m.

Consider the (t, ε)-plane in $\mathbb{P}$ through $\delta = 0$ and $m = 1$. The intersection of all regions A_G with this plane are the points (t, ε) for which G is a $(1, \varepsilon, 0)$-group. Note that G is a $(1, \varepsilon, 0)$-group at time t if and only if the set G is ε-connected at time t. A_G, restricted to this plane, is simply connected. Furthermore, as the distance between any pair of entities moving in $\mathbb{R}^1$ varies linearly, A_G is bounded from below by a t-monotone polyline f_G. The region is unbounded from above: if G is ε-connected (at time t) for some value ε,

*Department of Mathematics and Computer Science, TU Eindhoven, [a.i.v.goethem|b.speckmann]@tue.nl

†Dept. of Computing and Information Sciences, Utrecht University, The Netherlands, [m.j.vankreveld|m.loffler]@uu.nl

‡MADALGO, Aarhus University, Denmark, f.staals@cs.au.dk

then it is also ε'-connected for any $\varepsilon' \geq \varepsilon$ (see Fig. 1). Every maximal length segment in the intersection between (the restricted) A_G and the horizontal line ℓ_ε at height ε corresponds to a (maximal) time interval I during which (G, I) is a $(1, \varepsilon, 0)$-group, or an *ε-group* for short. Every such a segment corresponds to an *instance* of ε-group G.

Observation 1 *Set G is a maximal ε-group on I, iff the line segment $s_{\varepsilon,I} = \{(t, \varepsilon) \mid t \in I\}$ is a maximal length segment in A_G, and is not contained in A_H, for a supergroup $H \supset G$.*

Two instances of ε-group G may *merge*. Let v be a local maximum of f_G and $I_1 = [t_1, v_t]$ and $I_2 = [v_t, t_2]$ be two instances of group G meeting at v. At v_ε, the two instances G that are alive during $[t_1, v_t]$ and $[v_t, t_2]$ merge and we now have a single time interval $I = [t_1, t_2]$ on which G is a group. We say that I is a new instance of G, different from I_1 and I_2. We can thus decompose A_G into maximally-connected regions, each corresponding to a distinct instance of group G, using horizontal segments through the local maxima of f_G. We further split each region at the values ε where G changes between being maximal and being dominated. Let $\mathcal{P}_G$ denote the obtained set of regions in which G is maximal. Each such a region P corresponds to a *combinatorially distinct* instance on which G is a maximal group (with at least one member and duration at least zero). The region P is bounded by at most two horizontal line segments and two ε-monotone chains (see Fig. 1(b)).

Counting maximal ε-groups. To bound the number of distinct maximal ε-groups, over all values of ε, we count the number of polygons in $\mathcal{P}_G$ over all sets G. Consider a distinct instance (a set of entities G and a region $P \in \mathcal{P}_G$) of the maximal ε-group G. All vertices of P lie on the polyline f_G: they are either vertices of f_G, or they are points (t, ε) on the edges of f_G where G starts or stops being maximal. Any vertex is used by at most a constant number of regions from $\mathcal{P}_G$.

Below we show that the complexity of the arrangement $\mathcal{H}$, of all polylines f_G over all G, is bounded by $O(|\mathcal{A}|n)$. Furthermore, we show that each vertex of $\mathcal{H}$ can be incident to at most $O(n)$ regions. It follows that the complexity of all polygons $P \in \mathcal{P}_G$, over all groups (sets) G, and thus also the number of such sets, is at most $O(|\mathcal{A}|n^2)$.

The complexity of $\mathcal{H}$. The *span* $S_G(t) = \{a \mid a \in \mathcal{X} \ \wedge \ a(t) \in [\min_{b \in G} b(t), \max_{b \in G} b(t)]\}$ of a set of entities G at time t is the set of entities between the lowest and highest entity of G at time t. Let $h_a(t)$ denote the distance from entity a to the entity directly above a at time t, that is, $h_a(t)$ is the height of the face in $\mathcal{A}$ that has a on its lower boundary at time t.

Observation 2 *A set G is ε-connected at time t, if and only if the largest nearest neighbor distance among the entities in $S_G(t)$ is at most ε. Hence*

$$f_G(t) = \max_{a \in S_G(t)} h_a(t)$$

It follows that $\mathcal{H}$ is actually the arrangement of the n functions h_a, for $a \in \mathcal{X}$. We use this fact to show that $\mathcal{H}$ has complexity at most $O(|\mathcal{A}|n)$:

Lemma 1 *Let $\mathcal{A}$ be an arrangement of n line segments, and let k be the maximum number of line segments intersected by a vertical line. The number of triplets (F, F', x) such that the faces $F \in \mathcal{A}$ and $F' \in \mathcal{A}$ have equal height h at x-coordinate x is at most $O(|\mathcal{A}|k) \subseteq O(|\mathcal{A}|n) \subseteq O(n^3)$.*

Lemma 2 *The arrangement $\mathcal{H}$ has size $O(|\mathcal{A}|n)$.*

It remains to show that each vertex v of $\mathcal{H}$ can be incident to at most $O(n)$ polygons from different sets. Lemma 3 follows from Buchin et al. [4]:

Lemma 3 *Let $\mathcal{R}$ be the Reeb graph for a fixed value ε capturing the movement of a set of n entities moving along piecewise-linear trajectories in $\mathbb{R}^d$ (for some constant d), and let v be a vertex of $\mathcal{R}$. There are at most $O(n)$ maximal groups that start or end at v.*

Lemma 4 *Let v be a vertex of $\mathcal{H}$. Vertex v is incident to at most $O(n)$ polygons from $\mathcal{P} = \bigcup_{G \subseteq \mathcal{X}} \mathcal{P}_G$.*

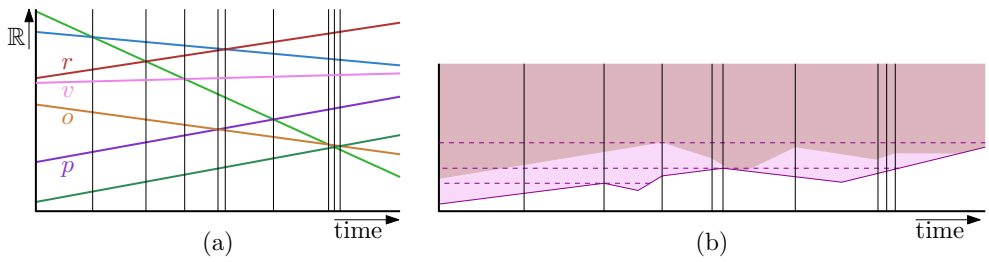


Figure 1: (a) A set of trajectories for a set of entities moving in $\mathbb{R}^1$ (b) The region $A_{\{r,v\}}$ during which $\{r, v\}$ is alive, and its decomposition into polygons, each corresponding to a distinct instance. In all such regions, except the top one $\{r, v\}$ is a maximal group: in the top region $\{r, v\}$ is dominated by $\{r, v, o\}$ (darker region).

Lemma 5 *The number of distinct ε-groups, over all values ε, and the total complexity of all regions $\mathcal{P} = \bigcup_{G \subseteq \mathcal{X}} \mathcal{P}_G$, are both at most $O(|\mathcal{H}|n) = O(|\mathcal{A}|n^2)$.*

Theorem 6 *Let $\mathcal{X}$ be a set of n entities, in which each entity travels along a piecewise-linear trajectory of τ edges in $\mathbb{R}^1$, and let $\mathcal{A}$ be the resulting trajectory arrangement. The number of distinct maximal groups is at most $O(|\mathcal{A}|n^2) = O(\tau n^4)$, and the total complexity of all regions in the parameter space corresponding to these groups is also $O(|\mathcal{A}|n^2) = O(\tau n^4)$.*

Lemma 7 *For a set $\mathcal{X}$ of n entities, in which each entity travels along a piecewise-linear trajectory of τ edges in $\mathbb{R}^1$, there can be $\Omega(\tau n^3)$ maximal ε-groups.*

3 Algorithm

We now refer to combinatorially different maximal groups simply as groups. Our algorithm computes a representation (of size $O(|\mathcal{A}|n^2)$) of all groups, which we can use to list all groups and, given a pointer to a group G, list all its members and the *grouping polygon* $Q_G \in \mathcal{P}_G$. We assume $\delta = 0$ and $m = 1$.

We use the arrangement $\mathcal{H}$ in the (t, ε)-plane. Line segments in $\mathcal{H}$ correspond to the height function of the faces in $\mathcal{A}$. Let $a, b \in S_G(t)$ be the pair of consecutive entities in the span of a group G with maximum vertical distance at time t. The *critical pair* (a, b) determines the minimal value of ε such that the group G is ε-connected at time t. The distance between (a, b) defines an edge of the polygon bounding G in $\mathcal{H}$.

Our representation consists of the arrangement $\mathcal{H}$ in which each edge e is annotated with a data structure $\mathcal{T}_e$, a list $\mathcal{L}$ with the top edge in each grouping polygon $Q_G \in \mathcal{P}_G$, and a data structure $\mathcal{S}$ to support reconstructing the grouping polygons.

We compute $\mathcal{H}$ in $O(|\mathcal{H}|) = O(\tau n^3)$ time [1]. Given $\mathcal{H}$ we use a sweep line algorithm to construct the representation. A horizontal line $\ell(\varepsilon)$ is swept at height ε upwards, and all groups G whose grouping polygon Q_G currently intersects ℓ are maintained. To achieve this we maintain a two-part status structure. First, a set $\mathcal{S}$ with for each group G the time interval $I(G, \varepsilon) = Q_G \cap \ell(\varepsilon)$. We can implement $\mathcal{S}$ using any standard balanced binary search tree. Second, for each edge $e \in \mathcal{H}$ intersected by $\ell(\varepsilon)$ a data structure $\mathcal{T}_e$ with the sets of entities whose time interval starts or ends at e, that is, $G \in \mathcal{T}_e$ if and only if $I(G, \varepsilon) = [s, t]$ with $s = e \cap \ell(\varepsilon)$ or $t = e \cap \ell(\varepsilon)$. The data structures $\mathcal{T}_e$ support the operations listed below.

In addition, we store with each interval $I(G, \varepsilon)$ a pointer to the previous version of the interval $I(G, \varepsilon')$ if (and only if) the starting time (ending time) of G changed to a different edge at ε'.

The data structure $\mathcal{T}_e$. We need a data structure $\mathcal{T} = \mathcal{T}_e$ that supports Filter, Insert, Delete, Merge, Contains, and HasSuperSet efficiently. We describe a structure of size $O(n)$, that supports Contains and HasSuperSet in $O(\log n)$ time, Filter in $O(n)$ time, and Insert and Delete in amortized $O(\log^2 n)$ time. In general, answering Contains and HasSuperSet queries in a dynamic setting is hard and may require $O(n^2)$ space [6].

Lemma 8 *Let G and H be two non-empty ε-groups that both end at time t. We have:*

$$(G \cap H \neq \emptyset \wedge |G| \leq |H|) \Longleftrightarrow G \subseteq H \wedge G \neq \emptyset.$$

We implement $\mathcal{T}$ with a tree similar to the *grouping-tree* used by Buchin et al. [4]. Let $\{G_1, .., G_k\}$ denote the groups stored in $\mathcal{T}$, and let $\mathcal{X}' = \bigcup_{i \in [1,..,k]} G_i$ denote the entities in these groups. Our tree $\mathcal{T}$ has a leaf

Operation	Input	Action
Filter$(\mathcal{T}_e, X)$	A data structure $\mathcal{T}_e$ A set of entities X	Create a data structure $\mathcal{T}' = \{G \cap X \mid G \in \mathcal{T}_e\}$
Insert$(\mathcal{T}_e, G)$	A data structure $\mathcal{T}_e$ A pointer to a representation of G	Create a data structure $\mathcal{T}' = \mathcal{T}_e \cup \{G\}$.
Delete$(\mathcal{T}_e, G)$	A data structure $\mathcal{T}_e$ A pointer to a representation of G	Create a data structure $\mathcal{T}' = \mathcal{T}_e \setminus \{G\}$.
Merge$(\mathcal{T}_e, \mathcal{T}_f)$	Two data structures $\mathcal{T}_e$, $\mathcal{T}_f$, belonging to two edges e, f having the same starting or ending vertex	Create a data structure $\mathcal{T}' = \mathcal{T}_e \cup \mathcal{T}_f$.
Contains$(\mathcal{T}_e, G)$	A data structure $\mathcal{T}_e$ A pointer to a representation of G ending or starting on edge e	Test if $\mathcal{T}_e$ contains set G.
HasSuperSet$(\mathcal{T}_e, G)$	A data structure $\mathcal{T}_e$ A pointer to a representation of G ending or starting on edge e	Test if $\mathcal{T}_e$ contains a set $H \supseteq G$, and return the smallest such set if so.

for every entity in $\mathcal{X}'$. Each group G_i is represented by an internal node v_i. For each internal node v_i the set of leaves in the subtree rooted at v_i corresponds exactly to the entities in G_i. By Lemma 8 these sets indeed form a tree. With each node v_i, we store the size of G_i, and an arbitrary entity in G_i. We preprocess $\mathcal{T}$ in $O(n)$ time to support level-ancestor (LA) queries as well as lowest common ancestor (LCA) queries, using the methods of Bender and Farach-Colton [2, 3]. Both methods work only for *static* trees, whereas we need updates to $\mathcal{T}$ as well. Since we query $\mathcal{T}_e$ only when processing the upper end vertex of e, we can be lazy in updating $\mathcal{T}_e$ and simply rebuild $\mathcal{T}_e$ when needed.

HasSuperSet and Contains queries. Using LA queries we can do a binary search on the ancestors of a given node. This allows us to implement both HasSuperSet$(\mathcal{T}_e, G)$ queries and Contains$(\mathcal{T}_e, G)$ in $O(\log n)$ time for a group G ending or starting on edge e. Let a be an arbitrary element from group G. If the data structure $\mathcal{T}_e$ contains a node matching the elements in G then it must be an ancestor of the leaf containing a in $\mathcal{T}$. That is, it is the ancestor that has exactly $|G|$ elements. By Lemma 8 there is at most one such node. As ancestors get only more elements as we move up the tree, we find this node in $O(\log n)$ time by binary search. Similarly, we can implement the HasSuperSet function in $O(\log n)$ time.

Insert, Delete, and Merge queries. The Insert, Delete, and Merge operations on $\mathcal{T}_e$ are performed lazily; we execute them only when we get to the upper vertex of edge e. At such a time we may have to process a batch of $O(n)$ such operations which we can handle in $O(n \log^2 n)$ time.

Lemma 9 *Let $G_1,..,G_m$ be maximal ε-groups, ordered by decreasing size, such that: (i) all groups end at time t, (ii) $G_1 \supseteq G_i$, for all i, (iii) the largest group G_1 has size s, and (iv) the smallest group has size $|G_m| > s/2$. We then have that $G_i \supseteq G_{i+1}$ for all $i \in [1,..,m-1]$.*

Lemma 10 *Given two nodes $v_G \in \mathcal{T}$ and $v_H \in \mathcal{T}'$, representing the set G respectively H, both ending at time t, we can test if $G \subseteq H$ in $O(1)$ time.*

Lemma 11 *Given $m = O(n)$ nodes representing maximal ε-groups $G_1,..,G_m$, possibly in different data structures $\mathcal{T}_1,..,\mathcal{T}_m$, that all share ending time t, we can construct a new data structure $\mathcal{T}$ representing $G_1,..,G_m$ in $O(n \log^2 n)$ time.*

The final function Filter can easily be implemented in linear time by pruning the tree from the bottom up.

Lemma 12 *We can handle each event in $O(n \log^2 n)$ time.*

Reconstructing the grouping polygons. Given a group G we can construct the complete grouping polygon Q_G in $O(|Q_G|)$ time, and list all group members of G in $O(|G|)$ time. We have access to the top edge of Q_G. This is an interval $I(G, \hat{\varepsilon})$ in $\mathcal{S}$, specifically, the version corresponding to $\hat{\varepsilon}$, where $\hat{\varepsilon}$ is the value at which G dies as a maximal group. We then follow the pointers to the previous versions of $I(G, \cdot)$ to construct the left and right chains of Q_G. When we encounter the value $\check{\varepsilon}$ at which G is born, these chains either meet at the same vertex, or we add the final bottom edge of Q_G connecting them. To report the group members of G, we follow the pointer to $I(G, \hat{\varepsilon})$ in $\mathcal{S}$. This interval stores a pointer to its starting edge e, and to a subtree in $\mathcal{T}_e$ of which the leaves represent the entities in G.

Analysis. The list $\mathcal{L}$ contains $O(g) = O(|\mathcal{A}|n^2)$ entries (Theorem 6), each of constant size. The total size of all $\mathcal{S}$'s is $O(|\mathcal{H}|n)$: at each vertex of $\mathcal{H}$, there are only a linear number of changes in the intervals in $\mathcal{S}$. Each edge e of $\mathcal{H}$ stores a data structure $\mathcal{T}_e$ of size $O(n)$. It follows that our representation uses a total of $O(|\mathcal{H}|n) = O(|\mathcal{A}|n^2)$ space. Handling each of the $O(|\mathcal{H}|)$ nodes requires $O(n \log^2 n)$ time, so the total running time is $O(|\mathcal{A}|n^2 \log^2 n)$.

Theorem 13 *Given a set $\mathcal{X}$ of n entities, in which each entity travels along a trajectory of τ edges, we can compute a representation of all $g = O(|\mathcal{A}|n^2)$ combinatorial maximal groups $\mathcal{G}$ such that for each group $G \in \mathcal{G}$ we can report its grouping polygon and its members in time linear in its complexity and size, respectively. The representation has size $O(|\mathcal{A}|n^2)$ and takes $O(|\mathcal{A}|n^2 \log^2 n)$ time to compute, where $|\mathcal{A}| = O(\tau n^2)$ is the complexity of the trajectory arrangement.*

References

[1] N. Amato, M. Goodrich, and E. Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In *Proc. 11th ACM-SIAM Symp. on Disc. Algorithms*, pages 705–706, 2000.

[2] M. Bender and M. Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoret. Informatics*, volume 1776 of *LNCS*, pages 88–94. Springer, 2000.

[3] M. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theoret. Computer Science*, 321(1):5–12, 2004.

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

[5] D. Eppstein, M. Goodrich, and J. Simons. Set-difference range queries. In *Proc. 2013 Canadian Conf. on Comput. Geom.*, 2013.

[6] D. Yellin. Representing sets with constant time equality testing. *J. of Algorithms*, 13(3):353–373, 1992.

New Results on Trajectory Grouping under Geodesic Distance*

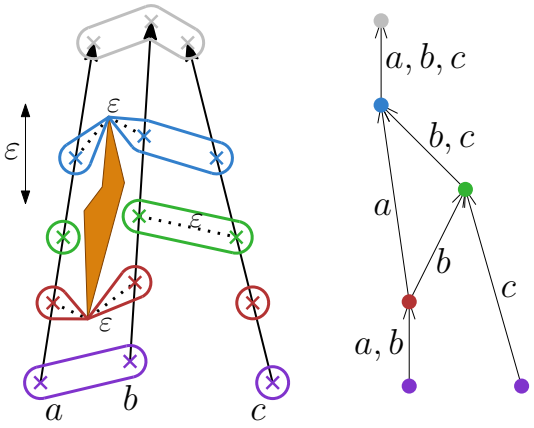
Maarten Löffler† Frank Staals‡ Jérôme Urhausen§

Abstract

We study grouping of entities moving amidst obstacles, extending the recent work of Kostitsyna et al. [5]. We present an alternative algorithm that can compute the Reeb-graph, a graph which captures when and how the partition of the entities into groups changes, when the entities move amidst arbitrary polygonal obstacles. Our new algorithm is significantly faster than the algorithm of Kostitsyna et al. when the number of entities is significantly larger than the total complexity of the obstacles. Furthermore, we consider a restricted setting in which the obstacles are big compared to ε: the parameter determining when entities are close enough together to be in the same group. We show that in this setting the Reeb-graph is much smaller, and we can compute it much faster, than in the case of general obstacles.

1 Introduction

In recent years, trajectory analysis has become a popular and well studied topic in computational geometry [1, 2, 3, 5]. We consider the problem of finding all (maximal) *groups* from the trajectory data. Intuitively, a group is a sufficiently large set of entities that travel together for a sufficiently long time. Buchin et al. [2] formalize this notion of groups, and show how to compute all *maximal* groups efficiently. A group is said to be maximal if the time interval on which the entities are together is maximal in length, and there is no group that contains it and stays together during the same time interval. Recently, Kostitsyna et al. [5] significantly extended the work of Buchin et al. by considering the environment in which the entities move. In particular, they study grouping when the entities move amidst various types of obstacles (see Table 1). So, when we decide if two entities are close enough together, we measure the distance using the geodesic distance (i.e. the length of the smallest obstacle-avoiding path) rather than the Euclidean distance. We continue the work of Kostitsyna et al. in two ways. First, we present an improved algorithm for the case in which the entities move amidst arbitrary obstacles, but their total complexity m is small compared to the number of entities n. Second, we consider a new environment setting, in which the obstacles are "large" (but may be arbitrarily complex and close together). This allows us to give a much faster algorithm than in the case of arbitrary obstacles. Next, we present the required notation and definitions following Buchin et al. [2] and Kostitsyna et al. [5], and formally define our problem, so that we can state our results more precisely.



(a) Entities a, b and c move along a linear trajectory around an obstacle. The colors of the entities indicate their positions at important moments. The circular forms indicate the groups at those moments.

(b) The Reeb graph for entities a, b, and c. The colors of the vertices correspond to the colors used in the figure on the left in order to indicate timestamps.

Figure 1: Example of entities moving in the two-dimensional space with obstacles and the corresponding Reed graph.

Notation and Problem Definition. We are given a set $\mathcal{X}$ of n entities, each moving along a piecewise linear trajectory with τ vertices, and a set of pairwise disjoint polygonal obstacles $\mathcal{O} = \{\mathcal{O}_1, .., \mathcal{O}_h\}$. Let m denote the total complexity of $\mathcal{O}$.

To determine if a set of entities may form a group, we have to decide if they are close together. We model this by a parameter ε. Two entities a and b are *directly connected* at time t if they are within geodesic distance ε from each other, that is, $\varsigma_{ab}(t) \leq \varepsilon$. A set of entities $\mathcal{X}'$ is *ε-connected* at time t if for any pair

*FS is supported by the Danish National Research Foundation under grant nr. DNRF84.

†Dept. of Information and Computing Sciences, Utrecht University, m.loffler@uu.nl

‡MADALGO, Aarhus University, f.staals@cs.au.dk

§Dept. of Informatics, KIT, jerome.urhausen@gmail.com

Simple polygon	$O(\tau n^2(\log^2 m + \log n) + m)$
Well-spaced obstacles	$O(\tau n^2 m \log n)$
General obstacles	$O(\tau n^2 m^2 \log n + m^2 \log m)$
2ε-big obstacles	$O(\tau(n^2(\log n + \log^2 m) + nm \log m))$
ε-big obstacles	$O(\tau n^2(\text{polylog}\, m + \log n) + m^2 \log m)$
General obstacles	$O(\tau(n^2 m + \lambda_4(n)m^3)(\log n + \log m))$

Table 1: The running time for computing the Reeb graph for various obstacle configurations. The top ones are from [5], the bottom ones are new. The $\lambda_s(n)$ term denotes the maximum length of a Davenport-Schinzel sequence of order s consisting of n symbols.

$a, b \in \mathcal{X}'$ there is a sequence $a = a_0, a_1, .., a_k = b$ such that a_i and a_{i+1} are directly connected. We refer to a time at which a and b become directly connected or disconnected as an *ε-event*. At such a time the distance between a and b is exactly ε. If an ε-event also connects or disconnects the maximal ε-connected set(s) containing a and b, it is a *critical event*. A (maximal) ε-connected set of entities $\mathcal{X}'$ is a *group* if it is ε-connected at any time t in a time interval of length at least δ, and it has at least a certain size.

The algorithm of Buchin et al. [2] proceeds in two phases. In the first phase, it computes the *Reeb-graph* $\mathcal{R}$ capturing the connectivity between the entities. In the second phase, it computes all maximal groups using only information in the Reeb-graph. So, once we compute $\mathcal{R}$, we can use the algorithm from Buchin et al. [2] to compute all maximal groups. An edge (u, v) in $\mathcal{R}$ corresponds to a maximal set of entities that is ε-connected during time interval $[t_u, t_v]$. The Reeb-graph has a vertex v at time t_v if (and only if) two maximal sets of ε-connected entities merge or split. A vertex corresponds uniquely to a critical event. See Fig. 1.

Results and Organisation. We start in Section 2 with the new algorithm for the case that the entities move amidst general obstacles. In Section 3 we formalize what it means for an obstacle to be ε-big, and show that if the obstacles are ε-big, the Reeb graph has low complexity. Furthermore, we show that we can compute the Reeb-graph efficiently in such a setting. Omitted proofs and details can be found in [8].

2 An Algorithm for General Obstacles

In this section we present an $O(\tau(n^2 m + \lambda_4(n)m^3)(\log n + \log m))$ time algorithm to compute the Reeb-graph when the entities move amidst arbitrary polygonal obstacles. This improves the algorithm of Kostitsyna et al. [5] if the total complexity m of the obstacles is $\omega(n^2/\lambda_4(n))$.

Most existing algorithms to compute the Reeb graph $\mathcal{R}$ first determine all ε-events, and use them to maintain the *entity-graph* while varying the time t. The entity-graph $G(t)$ at time t is the graph whose vertices are the entities, and whose edges connect two entities if and only if they are directly connected at time t. Clearly, $G(t)$ changes only at ε-events. The Reeb graph corresponds exactly to the changes in connected components in the entity-graph. That is, there is a critical event at time t if and only if the connected components in the entity graph change at time t. However, the number of critical events, and thus the size of $\mathcal{R}$, is much smaller than the number of ε-events [5]. Hence, we wish to reduce the number of ε-events that we have to consider.

The ER-graph. Our new algorithm will still use the idea of the entity-graph, but we add new vertices, corresponding to regions, that allow us to handle multiple ε-events at once. For our new graph, the *entity-region graph* (ER-graph), we still require that two entities are in the same connected component if and only if they are ε-connected.

The regions corresponding to the new vertices are built around the obstacles such that multiple ε-events involving entities in these regions only induce few critical events. To achieve that, we subdivide for each obstacle vertex v the area within geodesic ε-distance of v using the shortest path map originating at v [4].

Claim. We can further subdivide the shortest path maps into $O(m^2)$ regions such that

- each region has constant complexity,
- each region has (geodesic) diameter at most ε, and
- each entity enters and exits a region at most once per time step.

In the ER-graph we then only directly connect entities by an edge if they are closer than ε and can see each other, i.e. if the shortest path between them does not use an obstacle vertex. All other connections use region vertices. We connect a region to all the entities it contains and we connect two regions belonging to the same obstacle vertex v if and only if they contain entities for which the shortest path between them passing through v has length at most ε. See Fig. 2 for an example. The use of regions drastically reduces the number of events to handle.

There are $O(\tau n^2 m)$ events at which an edge between two entities appears or disappears in the ER-graph. For edges between an entity and a region there are $O(\tau n m^2)$ such events and for edges between two regions there are $O(\tau \lambda_4(n) m^3)$ such events.

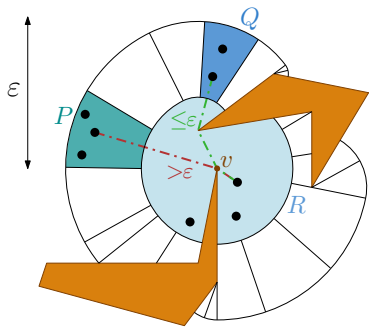


Figure 2: Entities inside regions corresponding to an obstacle vertex v at a fixed time t. Two shortest paths passing through v between entities in different regions are indicated in red and green. In the corresponding ER-graph the regions Q and R are connected by an edge, but P and R are not.

Algorithm The algorithm runs as follows. We first build the regions for each obstacle vertex. This gives us all the vertices of the ER-graph. Then we determine the events at which the edges of the ER-graph are added or removed and sort them. Using these events we keep an updated version of the ER-graph which allows us to build the Reeb graph. The regions can be built in $O(m^2 \log m)$ time. The determination and sorting of the events takes $O(\tau(n^2m + \lambda_4(n)m^3)\log(nm))$ time. In order to keep the connected components of the ER-graph updated we need $O(\tau(n^2m + \lambda_4(n)m^3)\log(n + m^2))$ time with the approach proposed by Parsa [7], because the ER-graph has at most $O(n + m^2)$ vertices.

Theorem 1 *Let $\mathcal{X}$ be a set of n entities, each moving amidst a set of obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The Reeb graph can be computed in $O(\tau(n^2m + \lambda_4(n)m^3)(\log n + \log m))$ time.*

3 Big Obstacles

In this section we investigate a new class of obstacles for which we can compute the Reeb graph efficiently. Namely, ε-big obstacles. An *ε-big obstacle* is an obstacle that does not fit into a strip of width ε of any orientation. We now show that if all obstacles are big there are only few ε-events, and thus the Reeb-graph is small, and we can compute it efficiently.

Two obstacle avoiding paths P_1 and P_2 have the same *homotopy type*, if and only if we can continuously deform P_1 into P_2 while remaining obstacle avoiding.

Lemma 2 *Let a and b be two entities moving amidst a set of ε-big obstacles. Let I be a time interval in which both a and b move linearly, and $t_1, t_2 \in I$ be two times at which their geodesic distance is at most ε. The geodesics $P_1 = \varsigma_{ab}(t_1)$ and $P_2 = \varsigma_{ab}(t_2)$ have the same homotopy type.*

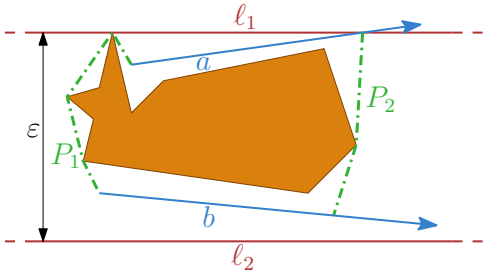


Figure 3: The construction showing that the obstacle can not be ε-big.

Proof. Assume, by contradiction, that P_1 and P_2 have different homotopy types. It follows that the region R bounded by $\overline{a(t_1)a(t_2)}$, P_2, $\overline{b(t_2)b(t_1)}$, and P_1 contains at least one obstacle. Let ℓ_1 be an outer tangent to P_1 and P_2 (See Fig. 3), and assume without loss of generality that ℓ_1 is horizontal, and that P_1 and P_2 lie below ℓ_1. Let ℓ_2 be the horizontal line at distance ε below ℓ_1.

We know that for $i = 1, 2$, every two points on P_i are at Euclidean distance at most ε, because the length of P_i is at most ε. Since both P_1 and P_2 have a common point with ℓ_1, all points on P_1 and P_2 lie on or above ℓ_2. It follows that $\overline{a(t_1)a(t_2)}$ and $\overline{b(t_2)b(t_1)}$ also lie on or above ℓ_2. This means that the strip between ℓ_1 and ℓ_2 contains the region R, and thus at least one ε-big obstacle. Contradiction. It follows that P_1 and P_2 have the same homotopy type. □

Theorem 3 *Let $\mathcal{X}$ be a set of n entities, each moving amidst a set of ε-big obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The number of ε-events, and thus the size of the Reeb-graph, is at most $O(\tau n^2)$.*

Proof. There are $O(n^2)$ pairs of entities and for each pair a, b there are $O(\tau)$ time intervals in which both of them move along a line with constant speed. Consider such an interval I, and let $t_1, t_2 \in I$ be two ε-events involving a and b. By Lemma 2, the geodesics $\varsigma_{ab}(t_1)$ and $\varsigma_{ab}(t_2)$ have the same homotopy type. Kostitsyna et al. [5] effectively show that if the homotopy type of the path between a and b is fixed, the length of such a path is a convex function in t, and thus, there are at most two ε-events involving a and b in interval I. The theorem then follows. □

3.1 Computing ε-events among 2ε-big obstacles

When all obstacles are 2ε-big we can efficiently compute the ε-events as follows. For each entity a and each interval I when a moves along an edge s, consider the geodesic *ε-surrounding* S of s, that is, all points

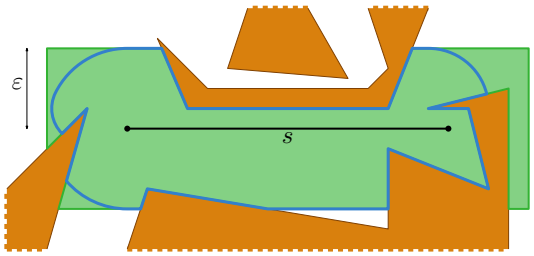


Figure 4: A simple polygon (green) containing the geodesic ε-surrounding (blue) of a trajectory edge s.

whose geodesic distance to s is at most ε. Clearly, all ε-events involving a in interval I are contained in S. Furthermore, since all obstacles are 2ε-big, S is simple; i.e. it contains no holes. See Fig. 4. We now construct a simple polygon P containing S, and compute ε-events involving a using the algorithm for entities moving in a simple polygon by Kostitsyna et al. [5].

We can construct P in $O(m \log m)$ time as follows. We first approximate the Euclidean ε-surrounding of s with the smallest rectangle possible. Then we calculate the intersections between the rectangle and the obstacle edges and sort them clockwise. Starting from the obstacle vertex that is the closest to s we can then walk along the boundary of the simple polygon P using these intersection points.

Since we have $O(\tau n)$ trajectory edges, building all these polygons takes $O(\tau n m \log m)$ time. Then, for each pair of entities and each time interval in which they travel at constant speed, we take the polygon of one of the entities and determine the interval during which the other entity is inside this polygon. Therefore computing each of the $O(\tau n^2)$ ε-events can then be made using parametric search in $O(\log^2 m)$ time per event [6]. Once we have determined and sorted all ε-events we can build the Reeb graph using $O(\log n)$ time per event. We conclude:

Theorem 4 *Let $\mathcal{X}$ be a set of n entities, each moving amidst a set of 2ε-big obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The Reeb graph can be computed in $O(\tau(n^2(\log n + \log^2 m) + nm \log m))$ time, where m is the total complexity of $\mathcal{O}$.*

3.2 Computing ε-events among ε-big obstacles

The main difference to the previous case is that an ε-big obstacle can be completely contained inside the Euclidean ε-surrounding of the segment. This means that the previously taken approach of building a polygon by approximating the Euclidean ε-surrounding yields a polygon with holes. Thus for a pair of entities we do not know the homotopy type of the shortest path yet. Therefore another approach is taken here.

The global idea of our approach is as follows. We compute all Euclidean ε-events, ignoring the obstacles. This gives us $O(\tau n^2)$ time intervals during which two entities, say a and b, are within Euclidean distance ε and move linearly. Any geodesic ε-event occurs within such intervals $[t_f, t_g]$. Furthermore, by Lemma 2 there are at most two such events per interval. Then the following claim holds.

Claim. For any time $t \in [t_f, t_g]$, there are only $O(1)$ choices for the first and last vertex on the geodesic between $a(t)$ and $b(t)$.

The algorithm tries all such pairs using the shortest path maps [4] to find the true geodesic at time t. Hence, for a given time t, we can test if the shortest path between $a(t)$ and $b(t)$ has length at most ε and if the derivative is positive or negative in $O(\text{polylog}\, m)$ time. This means that we can use parametric search to find the times at which the geodesic distance is exactly ε [5]. Overall, we conclude:

Theorem 5 *Let $\mathcal{X}$ be a set of n entities, each moving amidst a set of ε-big obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The Reeb graph can be computed in $O(\tau n^2(\text{polylog}\, m + \log n) + m^2 \log m)$ time, using $O(m^2)$ space, where m is the total complexity of $\mathcal{O}$.*

References

[1] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111–125, 2008.

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

[3] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of patterns in 2d trajectories of moving points. *Geoinformatica*, 11(2):195–215, 2007.

[4] J. Hershberger and S. Suri. An Optimal Algorithm for Euclidean Shortest Paths in the Plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[5] I. Kostitsyna, M. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. Trajectory grouping structure under geodesic distance. In *31st Int. Symp. on Comp. Geom.*, volume 34 of *LIPICS*, pages 674–688, 2015.

[6] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.

[7] S. Parsa. A deterministic O(m log m) time algorithm for the reeb graph. *Discrete & Computational Geometry*, 49(4):864–878, 2013.

[8] J. Urhausen. New results on trajectory grouping under geodesic distance. `https://fstaals.net/research/bachelor_thesis_jerome.pdf`, 2015.

A Refined Definition for Groups of Moving Entities and its Computation*

Marc van Kreveld† Maarten Löffler† Frank Staals‡ Lionov Wiratma†§

Abstract

We propose a refined definition of a group of moving entities which corresponds better to human intuition. We also present algorithms to compute all maximal groups from a set of moving entities.

1 Introduction

Nowadays, inexpensive modern devices with advanced tracking technologies make it easy to track movements of an entity. This has led to the availability of movement data for various types of moving entities (human, animals, vehicles, etc.). Since a tracking device typically returns a single location at each time stamp, each moving entity will be represented as a moving point. The data may consist of just one trajectory tracked over a period, or a whole collection of trajectories that are all tracked over a period. It is common to denote the number of trajectories (or moving entities) by n and the number of time stamps used for each by τ. Hence, the input size is $\Theta(\tau n)$.

To analyze moving object data, a number of methods have been developed in recent times. These methods may concern similarity analysis, clustering, outlier detection, segmentation, and various patterns that may emerge from the movement of the entities (for surveys see [3, 14]). These methods are often based on geometric algorithms, because the data is essentially spatial.

One particular type of pattern that has been well-studied is flocking [1, 4, 5]. Intuitively, a flock is a subset of the entities moving together (or simply being together) over a period of time. Other names for this and closely related concepts with slightly different definitions are herds [6], convoys [8], moving clusters [9], mobile groups [7], swarms [11], and groups [2]. The last of these defines a group in a simple and formal way. In [2] a model is introduced called the *trajectory grouping structure* which not only defines groups, but also the splitting of a group into subgroups and its opposite, merging. The algorithmic problem of reporting all maximal groups that occur in the trajectories is solved in $O(\tau n^3 + N)$ time, where $N = O(\tau n^4)$ is the output size (the summed size of all groups reported). The algorithm also considers times in between the τ time stamps where the locations are recorded as relevant. In between these time stamps, locations are inferred by linear interpolation over time.

In this paper we continue the study of such groups, but we propose a refined definition to the one in [2]. We motivate why it captures our intuition better and present algorithms to compute all maximal groups.

2 Problem Description

The definition of a group by Buchin et al. [2] relies on three parameters: one for distance between entities, one for the duration of a group, and one for the size of a group. We review their definitions next.

For a set of moving entities $\mathcal{X}$, two entities x and y are *directly ε-connected* at time t if the Euclidean distance between x and y is at most ε at time t, for some given $\varepsilon \geq 0$. Two entities x and y are *ε-connected in $\mathcal{X}$* if there is a sequence $x = x_0, ..., x_k = y$, with $\{x_0, ..., x_k\} \subseteq \mathcal{X}$ and for all i, x_i and x_{i+1} are directly ε-connected at the same time t.

In [2], a *group* for an entity inter-distance ε, a minimum required duration δ, and a minimum required size m, is defined as a subset $G \subseteq \mathcal{X}$ and corresponding time interval I for which three conditions hold:

- (i) G contains at least m entities.
- (ii) I has a duration at least δ.
- (iii) Every two entities $x, y \in G$ are ε-connected in $\mathcal{X}$ during I.

Furthermore, a group G with time interval I is *maximal* if there is no time interval I' properly containing I for which G is also a group, and there is no proper supergroup G' of G that is also a group during I [2].

One issue with this definition is that it does not correspond fully to our intuition. Two entities x and y may form a maximal group in an interval I even if they are always far apart, as long as there are always entities of $\mathcal{X}$ in between to make x and y ε-connected in $\mathcal{X}$. This can have counter-intuitive effects especially in dense crowds. To avoid such issues, we refine the definition of a group. In particular, we replace condition (iii) above by:

- (iii') Every two entities $x, y \in G$ are ε-connected in G during I.

*M.L. supported by the Netherlands Organisation for Scientific Research (NWO, grant 639.021.123). F.S. supported by the Danish National Research Foundation (grant nr. DNRF84). L.W. supported by the Directorate General of Resources for Science, Technology and Higher Education of Indonesia.

†Dept. of Information and Computing Sciences, Utrecht University, {m.j.vankreveld,m.loffler,l.wiratma}@uu.nl

‡MADALGO, Aarhus University, f.staals@cs.au.dk

§Dept. of Informatics, Parahyangan Catholic University

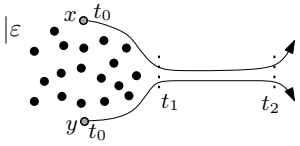


Figure 1: In the definition by [2], x and y are ε-connected during $[t_0, t_2]$

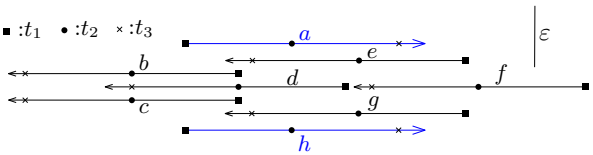


Figure 2: Entities in $G = \{a, h\}$ are ε-connected using entities not in G.

We define maximal groups in the same way as before.

We give two examples that show the difference in these definitions. First, consider a number of stationary entities S and two entities x and y, see Figure 1.

Entity x starts (t_0) to the North of S and moves around its perimeter to the East. Entity y starts (t_0) to the South and also moves around the perimeter to the East. After encountering (t_1) each other at the East side, both continue together eastward, away from the stationary entities in S (ending at t_2).

By the definition in [2], x and y form a maximal group in the interval $[t_0, t_2]$. In our refined definition, they form a maximal group during $[t_1, t_2]$, starting when x and y actually encounter each other.

Second, the previous definition can even see groups of entities that were never close, see Figure 2. Here, $\{a, h\}$ is a maximal group in the interval $I = [t_1, t_3]$ using the definition in [2]. At each time, a and h are ε-connected, but through different subsets of entities. Although a and h move in the same direction with the same speed, intuitively they do not form a group because they are too far apart and separated by other entities that move in the opposite direction. With the new, refined definition, we do not consider $\{a, h\}$ a group in the interval I.

Results and Organization. In this paper, we show that for a set $\mathcal{X}$ of n moving entities in $\mathbb{R}^1$ with τ time stamps each, the number of maximal groups by the refined definition is $O(\tau n^3)$, which is tight in the worst case. We present algorithms to compute all maximal groups, beginning with a basic algorithm that runs in $O(\tau^3 n^6)$ time. Subsequent improvements lead to a running time of $O(\tau^2 n^4)$. For moving entities in $\mathbb{R}^d$ $(d > 1)$, we show that all maximal groups can be computed in $O(\tau^2 n^5 \log n)$ time. From now on, we will use the term "group" to denote a group of entities that comply with our refined definition.

3 Preliminaries

Let $\mathcal{X}$ be a set of n entities moving in $\mathbb{R}^1$, given by locations at τ time stamps. A trajectory of an entity in $\mathcal{X}$ can be expressed by a piecewise-linear function which maps time to a point in $\mathbb{R}^1$. If $\mathbb{R}^1$ is associated with the vertical axis and time with the horizontal axis of a 2-dimensional plane, the trajectories of entities in $\mathcal{X}$ are polylines with τ vertices each. We will use the same notation to denote an entity and its trajectory. We assume that all trajectories have their vertices at the same times and that there are no two parallel edges.

Let $d_{ij}(t)$ be the Euclidean distance between $i \in \mathcal{X}$ and $j \in \mathcal{X}$ at time t. When $d_{ij}(t) = \varepsilon$, we say that an *ε-event* occurs. For any ε-event v, we denote by t_v the time when v occurs and $\omega(v)$ the function that returns the two entities that create v. We assume that no two or more ε-events occur at the same time.

Consider an ε-event v; let $\omega(v) = \{i, j\}$. If i and j are further than ε immediately before t_v, then v is a *start ε-event*; if they are further immediately after t_v it is an *end ε-event*. If there is no entity $k \in \mathcal{X}$ located strictly in between i and j at t_v (so $d_{ik}(t_v) + d_{jk}(t_v) = \varepsilon$), then we say that v is a *free ε-event*.

Observation 1 *The number of ε-events is $O(\tau n^2)$.*

Let G be a group of entities in time interval I that is maximal in size. All entities in G are pairwise ε-connected in the interval I, and hence, there are no free ε-events in G during I. In the arrangement of trajectories from G, no face has height greater than ε.

It is also clear that G can begin only at a start ε-event and end only at an end ε-event. Furthermore, we observe that if a start ε-event (or end ε-event) of G is not a free ε-event with respect to the entities in G, then before (or after) the interval I, entities in G are still pairwise ε-connected and we can extend the interval of G. Therefore, G can be a maximal group only if both the start ε-event and end ε-event are free ε-events (but this is not a sufficient condition).

Observation 2 *There can be at most one maximal group that starts and ends at a particular pair of start ε-event and end ε-event.*

Theorem 1 *For a set $\mathcal{X}$ of n entities, each entity moving along a piecewise-linear trajectory of τ edges, the maximum number of maximal group is $\Theta(\tau n^3)$.*

Proof. Any group G that starts at a start ε-event contains at most n entities. When a free end ε-event involving G occurs, only group G ends but a subgroup of G with fewer entites may continue. This can happen at most $n - 1$ times. Therefore, the number of maximal groups is $O(\tau n^3)$. Furthermore, there can be $\Omega(\tau n^3)$ maximal groups because the lower bound for the definition of a group in [2] still applies [13]. □

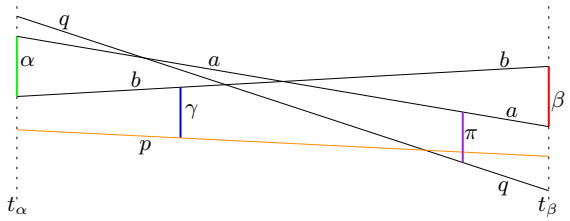


Figure 3: Removing trajectory p (and γ) introduces a new free ε-event: π.

The approach to compute all maximal groups is to work on the arrangement $\mathcal{A}$ of line segments that are the trajectories. For a subset $G \subseteq \mathcal{X}$ and interval I, we can remove entities from G that are separated at a face with height larger than ε in I (corresponding to a free ε-event). Only if there are no such faces, the remaining entities in G can be a group. Note that removing entities in G involves removing the corresponding trajectories from the arrangement $\mathcal{A}$, which can cause new faces that are free ε-events.

4 Basic Algorithm

Next, we describe a simple algorithm to compute all maximal groups. Let ξ_s and ξ_e be the sets of all start ε-events and all end ε-events respectively. Fix $\alpha \in \xi_s$ and $\beta \in \xi_e$. By Observation 2, there is only one maximal group G that starts at α and ends at β. Furthermore, observe that G necessarily contains the entities $\omega(\alpha) = \{a, b\}$ and $\omega(\beta) = \{c, d\}$, and that if G is a maximal group on $I = [t_\alpha, t_\beta]$, then all entities in G are on the same side at time $t_\gamma \in (t_\alpha, t_\beta)$ when a free ε-event γ occurs. We then use the following approach to find G (if it exists):

1. Initialize a set G containing all entities in $\mathcal{X}$.
2. Build an arrangement $\mathcal{A}$ induced by the trajectories of the entities in G in the interval I.
3. A face f in $\mathcal{A}$ contains a free ε-event γ if (and only if) the height of f is more than ε. If f has height larger than ε, test if (the trajectories of) a, b, c, and d, all lie on the same side of f. If not, there is no maximal group G that starts at α and ends at β. If they do pass on the same side, let S denote the set of entities whose trajectories lie on the other side of f. Remove these entities from S, and remove their trajectories from $\mathcal{A}$. Observe that new free ε-events may appear because removal of a trajectory from $\mathcal{A}$ merges two faces of $\mathcal{A}$ into a larger one. See Figure 3. Repeat this step until there is no more free ε-events γ with $t_\gamma \in (t_\alpha, t_\beta)$.
4. Check that α and β are now free. If so, G is a maximal group on I, and hence we can report it. If not, G is actually a group during a time interval $I' \supset I$. Hence, G may be maximal in size, but not in duration. We do not report G in this case.

Theorem 2 *Given a set $\mathcal{X}$ of n entities in which each entity moves in $\mathbb{R}^1$ along a trajectory of τ edges, all maximal groups can be computed in $O(\tau^3 n^6)$ time using the Basic Algorithm.*

Proof. The number of combination of a pair of start and end ε-events is $O(\tau^2 n^4)$. Building an arrangement from trajectories of entities takes $O(\tau n^2)$ time. Removing a trajectory e and checking new faces in $\mathcal{A}$ takes time proportional to the zone complexity of e: $O(\tau n)$. Since there are at most n trajectories to be removed, the whole process to remove entities for each interval I takes $O(\tau n^2)$ time. Therefore, the running time of the algorithm is $O(\tau^3 n^6)$ time. □

5 Improved Algorithm

The previous algorithm checks every pair of possible start and end ε-events α and β to potentially find one maximal group. To improve the running time, we fix a start ε-event α and consider the $O(\tau n^2)$ end ε-events β in increasing order. We show that we can check for a maximal group on $[t_\alpha, t_\beta]$ in amortized $O(1)$ time.

We build the arrangement $\mathcal{A}$ for all trajectories, starting from time t_α, and sort the end ε-events β, with $t_\beta > t_\alpha$ on increasing time. We then consider the end ε-events β in this order, while maintaining a maximal set G that is ε-connected in G throughout the time interval $[t_\alpha, t_\beta]$.

Let $\omega(\alpha) = \{a, b\}$ be the entities defining the start ε-event α, and let $G \supseteq \{a, b\}$ be the largest ε-connected set on $[t_\alpha, t_\beta]$. We compute the largest ε-connected set on $[t_\alpha, t_{\beta'}]$ for the next ending event β' as follows. Note that this set will be a subset of G.

Let S be the set of entities that separate from a and b at β. We remove all trajectories from the entities in S from $\mathcal{A}$. As before, this may introduce faces of height larger than ε. For every such face f, we check if a and b still pass f on the same side. If not, there can be no maximal groups that contain a and b, start at t_α, and end after t_β. If a and b lie on the same side of f, we add all entities that lie on the other side of f to S and remove their trajectories from $\mathcal{A}$. We repeat this until all faces in $\mathcal{A}$ that have non-empty intersection with the vertical strip defined by $[t_\alpha, t_{\beta'}]$ have height at most ε (or until we have found a face that splits a and b). It follows that the set $G' = G \setminus S$ is the largest set containing a and b that is ε-connected throughout $[t_\alpha, t_{\beta'}]$. If α and β' are free with respect to G' then we report G' as a maximal group.

Building the arrangement $\mathcal{A}$ takes $O(\tau n^2)$ time, and sorting the ending-events takes $O(\tau n^2 \log(\tau n))$ time. By the Zone Theorem, we can remove each trajectory in $O(\tau n)$ time. Checking the height of the new faces can be done in the same time bound. It follows that the total running time is $O(\tau n^2(\tau n^2 + \tau n^2 \log(\tau n) + R))$ where R is the total time for removing trajectories

from the arrangement. Clearly, R is bounded by the complexity of the arrangement: $O(\tau n^2)$. So, the total running time is $O(\tau^2 n^4 \log(\tau n))$.

Further Improvement We can avoid repeated sorting of end ε-events by pre-sorting them in a list, and for each start ε-event, use this list. The list will contain events that do not concern the entities involved in the start ε-event, but this can be tested easily in constant time. Thus, we conclude:

Theorem 3 *Given a set $\mathcal{X}$ of n entities in which each entity moves in $\mathbb{R}^1$ along a trajectory of τ edges, all maximal groups can be computed in $O(\tau^2 n^4)$ time.*

6 Algorithms for Entities in $\mathbb{R}^d$

In $\mathbb{R}^d$ ($d > 1$), it is harder to test whether an ε-event really connects or disconnects because the two entities may be ε-connected through other entities in the group. This observation immediately gives the condition for an ε-event to be *free*. We model our moving entities in a graph where vertices represent entities and an edge exists if two entities are directly ε-connected. As in Parsa [12], we can maintain the graph under edge updates, while allowing same component queries, in $O(\log n)$ time per operation.

To compute maximal groups, we start at a start ε-event α ($\omega(\alpha) = \{a, b\}$) and maintain the connected component $\mathcal{C}$ throughout the sequence of sorted ε-events. At each ε-event β, we remove any vertices that are disconnected from $\mathcal{C}$ and start again from α in case we remove anything. We stop if a and b are disconnected. If α is a free ε-event when we reach β again, we report $\mathcal{C}$ as a maximal group and continue.

We start at $O(\tau n^2)$ ε-events, process $O(\tau n^2)$ ε-events for each, and may need to restart up to $n - 1$ times. Hence, we obtain:

Theorem 4 *Given a set $\mathcal{X}$ of n entities move in $\mathbb{R}^d$ along a trajectory of τ edges, all maximal groups can be computed in $O(\tau^2 n^5 \log n)$ time.*

7 Conclusions and Future Work

In this paper we introduced a variation on the grouping structure definition [2] and argued that it corresponds better to our intuition. We have given an algorithm for trajectories moving in $\mathbb{R}^1$ that computes all maximal groups and runs in $O(\tau^2 n^4)$ time. In $\mathbb{R}^d$, our algorithm runs in $O(\tau^2 n^5 \log n)$ time. The number of maximal groups is $\Theta(\tau n^3)$ in the worst case.

The main challenges include reducing the dependency on τ to subquadratic, and the dependency on n. It would also be interesting to develop an output-sensitive algorithm that uses considerably less time if the output is small. Finally, we may be able to develop algorithms that take geodesic distance into account, like was done for the previous definition of a group [10].

References

[1] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111 – 125, 2008.

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

[3] J. Gudmundsson, P. Laube, and T. Wolle. Computational movement analysis. In *Handbook of Geographic Information*, pages 423–438. Springer, 2012.

[4] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *Proc. 14th ACM International Symposium on Advances in Geographic Information Systems*, GIS '06, pages 35–42, 2006.

[5] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica*, 11:195–215, 2007.

[6] Y. Huang, C. Chen, and P. Dong. Modeling herds and their evolvements from trajectory data. In *Geographic Information Science*, volume 5266 of *LNCS*, pages 90–105. Springer, 2008.

[7] S. Hwang, Y. Liu, J. Chiu, and E. Lim. Mining mobile group patterns: A trajectory-based approach. In *Advances in Knowledge Discovery and Data Mining*, volume 3518 of *LNCS*, pages 145–146. Springer, 2005.

[8] H. Jeung, M. Yiu, X. Zhou, C. Jensen, and H. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1:1068–1080, 2008.

[9] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Advances in Spatial and Temporal Databases*, volume 3633 of *LNCS*, pages 364–381. Springer, 2005.

[10] I. Kostitsyna, M. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. Trajectory grouping structure under geodesic distance. In *Proc. 31st International Symposium on Computational Geometry, SoCG 2015*, pages 674–688, 2015.

[11] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.

[12] S. Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. In *Proc. 28th Annual Symposium on Computational Geometry*, SoCG '12, pages 269–276, 2012.

[13] A. van Goethem, M. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. Grouping time-varying data for interactive exploration. In *Proc. 32th Annual Symposium on Computational Geometry*, SoCG '16, 2016.

[14] Y. Zheng and X. Zhou. *Computing with Spatial Trajectories*. Springer, 2011.

Fine-Grained Analysis of Problems on Curves

Kevin Buchin[1] Maike Buchin[2] Maximilian Konzack[3] Wolfgang Mulzer[4] André Schulz[5]

Abstract

We provide conditional lower bounds on two problems on polygonal curves. First, we generalize a recent result on the (discrete) Fréchet distance to k curves. Specifically, we show that, assuming the Strong Exponential Time Hypothesis, the Fréchet distance between k polygonal curves in the plane with n edges cannot be computed in $O(n^{k-\varepsilon})$ time, for any $\varepsilon > 0$. Our second construction shows that under the same assumption a polygonal curve with n edges in dimension $\Omega(\log n)$ cannot be simplified optimally in $O(n^{2-\varepsilon})$ time.

1 Introduction

The *fine-grained complexity* of the (discrete) Fréchet distance between two curves has recently attracted a lot of attention. After a long period without major progress, Agarwal et al. devised a subquadratic $O\left(\frac{mn \log\log n}{\log n}\right)$-time algorithm for the discrete Fréchet distance on the word RAM [2]. Buchin et al. [10] gave a randomized algorithm for the continuous Fréchet distance with a running time slightly better than the classic bound of $O(n^2 \log n)$ [4]. Answering a question by Buchin et al. [10], Bringmann [6] showed that the (discrete) Fréchet distance cannot be computed in $O(n^{2-\varepsilon})$ time, for any $\varepsilon > 0$, assuming the *Strong Exponential Time Hypothesis* (SETH). This result was later refined and extended [8]. SETH states that for every $\varepsilon > 0$, there is a $k \in \mathbb{N}$ such that the satisfiability problem on k-CNF formulas with n variables and m clauses cannot be solved in time $m^{O(1)}2^{(1-\varepsilon)n}$.

Bringmann's work [6] triggered a lot of activity, leading to new conditional lower bounds for famous problems such as edit distance, dynamic-time warping, or longest common subsequence (LCS) [1,7]. For LCS, a more general bound states the non-existence of a $O(n^{k-\varepsilon})$-time algorithm for k strings over an alphabet of size $O(k)$. Our first result generalizes the lower bound on the discrete Fréchet distance to k curves.

Theorem 1 *For any $\varepsilon > 0$, the discrete Fréchet distance of k planar point sequences of length n cannot be decided in $O(n^{k-\varepsilon})$ time, unless SETH fails.*

The Fréchet distance between k curves was considered previously by Rote and Dumitrescu [12], who provide a near-quadratic time 2-approximation for the problem. Measuring the distance and analyzing the similarity between a set of parameterized curves in this way is also relevant for movement data analysis. For instance, it can be used in the analysis of collective movement, e.g., within a flock of birds, or to detect clusters of similar movement sequences [9].

Our second result is on simplifying a d-dimensional polygonal curve $\langle a_0, \ldots, a_n \rangle$. We consider the common variant [13] where the vertices of the simplified curve should be an ordered subsequence of the original vertices, and if a_i and a_j are consecutive in the simplification, then the distance between the subcurve $\langle a_i, a_{i+1}, \ldots, a_j \rangle$ and the line segment $a_i a_j$ should be at most a given $\varepsilon > 0$. We focus on the Hausdorff distance, although the reduction also applies to the Fréchet distance. There are two common variants of the simplification problem: *min-#*, in which ε is given and the number of vertices is to be minimized, and *min-ε* in which an upper bound on the number of vertices is given and ε is to be minimized.

Algorithms for the min-ε and the min-# problems with running time $O(n^2 \log n)$ and $O(n^2)$, respectively, are known for polygonal curves in the plane [11]. For the L_1-metric, Agarwal and Varadarajan [3] presented an $O(n^{4/3+\varepsilon})$-time algorithm. For curves in $\mathbb{R}^d$, Barequet et al. [5] developed efficient algorithms. Their algorithms run in near-quadratic time for $d = 3$ and in subcubic time for $d = 4$. If distance is measured according to the L_1- or the L_∞-metric, they achieve a running time of $O(n^2)$ and $O(n^2 \log n)$ for min-ε and min-#, respectively, in any fixed dimension. In particular, for L_∞ the dependency on the dimension is only a small-degree polynomial. It is a long-standing open problem whether the (near-)quadratic running time can be improved for the Euclidean distance [3].[1]

We show that, at least in sufficiently high (non-constant) dimension, this is not possible unless SETH fails. For L_∞, our construction shows that the algorithm by Barequet et al. is essentially optimal in high dimensions, assuming SETH.

Theorem 2 *There is no algorithm that optimally, min-# or min-ε, simplifies a polygonal curve with n edges in $\mathbb{R}^d$ with $d = \Omega(\log n)$ using ε-tolerance regions*

[1]TU Eindhoven, the Netherlands k.a.buchin@tue.nl
[2]RU Bochum, Germany maike.buchin@rub.de
[3]TU Eindhoven, the Netherlands m.p.konzack@tue.nl
[4]FU Berlin, Germany mulzer@inf.fu-berlin.de
[5]FeU Hagen, Germany andre.schulz@fernuni-hagen.de

[1]See also http://cs.smith.edu/~orourke/TOPP/P24.html.

in the L_1-, L_2- or L_∞-metric that runs in $O(n^{2-\varepsilon})$ time, for any $\varepsilon > 0$, unless SETH fails.

To prove the lower bounds, we use a reduction from the k-Orthogonal Vectors problem (as stated in [1]), using the notation $[n] := \{1, \ldots, n\}$.

Definition 1 (k-Orthogonal-Vectors) *Suppose we are given k lists $\{\alpha_i^1\}_{i\in[n]}$, $\{\alpha_i^2\}_{i\in[n]}$, …, $\{\alpha_i^k\}_{i\in[n]}$ of vectors in $\{0,1\}^d$. We need to decide whether there are k vectors $\alpha_{i_1}^1$, $\alpha_{i_2}^2$, …, $\alpha_{i_k}^k$ with $\sum_{h=1}^{d}\prod_{t\in[k]}\alpha_{i_t}^t[h] = 0$. Any such collection of vectors is called* orthogonal.

The following lemma is well known [1, 14].

Lemma 3 *If there is an $\varepsilon > 0$ such that k-Orthogonal Vectors on n vectors in $\{0,1\}^d$ with $d = \Omega(\log n)$ can be solved in $O(n^{k-\varepsilon})$ time, then SETH is false.*

2 Fréchet distance between k curves

We show the lower bound on the discrete Fréchet distance between k curves by a reduction from the k-Orthogonal Vectors problem. We begin with some notation. Let $A_1, \ldots, A_k$ be k sequences of points in the plane, $A_i = \langle a_1^i, \ldots, a_{n_i}^i\rangle$. By $a_j^i[h]$, for $h = 1, 2$, we denote the h-th coordinate of a_j^i. We set $S = [n_1] \times [n_2] \times \cdots \times [n_k]$.

We define a *coupling* of length m on S as a sequence $\mathcal{C} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_m\rangle$ such that we have $\mathcal{C}_i \in S$, $\mathcal{C}_1 = (0, 0, \ldots, 0)$, $\mathcal{C}_m = (n_1, n_2, \ldots, n_k)$, and $\mathcal{C}_{i+1}[h] = \mathcal{C}_i[h]$ or $\mathcal{C}_{i+1}[h] = \mathcal{C}_i[h] + 1$, for all $i = 0, \ldots, m-1$ and $h = 1, \ldots, k$. A coupling $\mathcal{C}$ defines an alignment of the curves $A_1, \ldots, A_k$, and we define the *coupled distance* as $d_{\mathcal{C}}(A_1, \ldots, A_k) := \max\left\{d(a^h_{\mathcal{C}_i[h]}, a^{h'}_{\mathcal{C}_i[h']}) \mid 0 \le i \le m, 1 \le h, h' \le k\right\}$, where d denotes the Euclidean distance. The *discrete Fréchet distance* $d_F(A_1, \ldots, A_k)$ between the k curves is the minimal coupled distance over all possible couplings.

Next, we describe our reduction. Suppose we have k lists $\{\beta_i\}_{i\in[n]}$, $\{\alpha_i^t\}_{i\in[n]}$, $t \in [k-1]$, of vectors $\alpha_i^t, \beta_i \in \{0,1\}^d$. We construct k curves $B, A^1, A^2, \ldots, A^{k-1}$. Their discrete Fréchet distance will be 1 if the given vector lists contain a collection of k orthogonal vectors, and strictly larger than 1, otherwise. The coordinates of the vectors are encoded by *coordinate gadgets* (CG), see Figure 1. Set $\delta := 1/100$, and for $i = 1, \ldots, k-1$, let $CG_i(0) := \langle(-0.5-\delta, 0), (0.5, 0), (-0.5-\delta, 0), \ldots, (0.5, 0), (-0.5-\delta, 0)\rangle$ be a curve with $2k-1$ vertices. We define $CG_i(1)$ to have the same vertices as $CG_i(0)$, except that the $2i$-th vertex is replaced by $(0.5+\delta, 0)$. Further we define $CG_B(0) := \langle(-0.5, 0), (0.5, 0), (-0.5, 0), \ldots, (0.5, 0), (-0.5, 0)\rangle$ with $2k-1$ vertices and $CG_B(1)$ in the same way but with only $2k-3$ vertices. We call the vertices at $(0.5, 0)$ *short spikes* and at $(0.5+\delta, 0)$ *long spikes*.

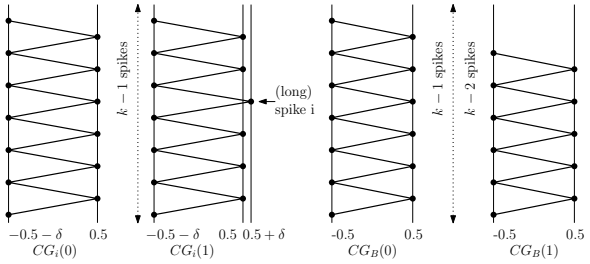


Figure 1: Coordinate gadgets (distorted vertically for the purpose of illustration).

Suppose that there were a coupling of $CG_1(1)$, $CG_2(1), \ldots, CG_{k-1}(1)$, $CG_B(1)$ achieving a distance of at most 1. Then, $k-1$ spikes of $CG_1(1)$,…, $CG_{k-1}(1)$ need to be coupled, but there is one long spike in each coupled spike. We need to couple every long spike with a different spike of $CG_B(1)$. This is not possible, since $CG_B(1)$ has only $k-2$ spikes. Thus, $d_F(CG_1(1), \ldots, CG_B(1)) > 1$. If we replace any $CG_*(1)$ with a respective curve $CG_*(0)$, the distance becomes 1.

Next, we encode the vectors and the vector lists. To "synchronize" coordinates, we will use the point $c := (0, 0.8661)$. The start of vectors will be demarcated by $v_A := (-0.499, -1)$ and $v_B := (0, -0.8661)$. Additionally, we will use the points $t_A = (0.48, -0.01)$ and $t_B = (0.57, 1.005)$ to mark a successful synchronized traversal, and $s = (-0.499, 0)$ as a point that is close to all except t_B, see Figure 2.

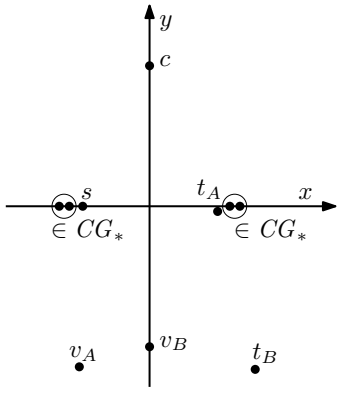


Figure 2: The points used as vertices of the curves.

Two points are *close* if their distance is at most 1: s is close to all points except t_B, and t_A is close to all except v_A. The point c is close only to s and t_A (and itself); t_B is close only to t_A and v_B; v_A is close only to s and v_B; v_B only to s, t_A and t_B.

Let $A_i^j := s \circ v_A \circ \bigcirc_{h=1}^{d}(CG_j(\alpha_i^j[h]) \circ c) \circ t_A$, where we use $\circ$ to denote the operation of adding a vertex

to a curve or of concatenating curves. We set $A^j := \left(\bigcirc_{i=1}^{n} A_i^j\right) \circ s$. Furthermore, we define $B_i := v_B \circ \bigcirc_{h=1}^{d}(CG_j(\beta_i[h]) \circ c)$ and $B := s \circ v_A \circ \bigcirc_{i=1}^{n} B_i \circ t_B \circ s$.

First, we argue that k vectors $\alpha_{i_1}^1, \alpha_{i_2}^2, \ldots, \alpha_{i_{k-1}}^{k-1}, \beta_{i_k}$ are orthogonal if and only if the corresponding concatenated coordinate gadgets have Fréchet distance at most 1. If the vectors are orthogonal, then in each coordinate, at least one vector has a 0-entry, and a coupling of distance at most 1 is possible. On the other hand, if the vectors are not orthogonal, there is one coordinate in which all vectors have 1-entries. The c vertices force us to traverse all coordinates simultenously, so that we will have to couple k one-coordinate gadgets, giving a Fréchet distance larger than 1.

Now, let us consider the vector lists and the complete curves. If the vector lists contain a k-tuple $\alpha_{i_1}^1, \alpha_{i_2}^2, \ldots, \alpha_{i_{k-1}}^{k-1}, \beta_{i_k}$ of orthogonal vectors, then the curves $A^1, \ldots, A^{k-1}, B$ have Fréchet distance at most 1. This can be seen by the following coupling: first, A^1 walks to the first point s of $A_{i_1}^1$, while all other curves wait at s. Then, A^2 walks to the first point s of $A_{i_2}^2$, while all other curves wait at s, etc. Finally, B walks to the first point v_B of B_{i_k}, while all other curves wait at s. Since s is close to all points except for t_B, the distance so far is 1. Then, the A^j curves simultaneously jump to v_A while B waits at v_B, and then the coordinate gadgets are traversed simultaneously. Next, the A^j curves wait at t_A while B walks to the last point s. The A^j then simultaneously go to the next s, and finish the walk to the final vertex one after another while the other curves wait at s.

Next, suppose that the curves have Fréchet distance larger than 1. We argue that then there is a k-tuple of orthogonal vectors. Indeed, suppose that no such k-tuple exists, and consider the first time that B reaches t_B. Since t_B is close only to t_A and v_B, at this point, all A^j must be at t_A. It follows that before that, all A^j's must have been simultaneously at v_A, because on the A^j's, v_A comes before t_A, and v_A is close only to s and v_B. For the same reason, at this point, B also must be at v_B. Then, the coordinate gadgets of a k-tuple of vectors are traversed simultaneously, leading to Fréchet distance larger than 1, as all k-tuples are non-orthogonal. Theorem 1 follows.

Our construction also rules out a faster polynomial-time approximation scheme unless SETH fails. The coordinates were computed by hand and could be optimized to prove a specific approximation lower bound.

3 Curve Simplification

In this section, we reduce the 2-Orthogonal Vectors problem to the curve simplification problem. Given two lists of 0/1-vectors $\{\alpha_i\}_{i\in[n]}$ and $\{\beta_i\}_{i\in[n]}$ in dimension d, we interpret each vector as a point in dimensions $d+1$, as follows: we define $\hat{\alpha}_i[h] := \alpha_i[h]$ for $1 \le h \le d$ and $\hat{\alpha}_i[d+1] := -\delta$ with $\delta = 2d^2$. We define $\hat{\beta}_i[h]$ analogously, except that $\hat{\beta}_i[d+1] := \delta$.

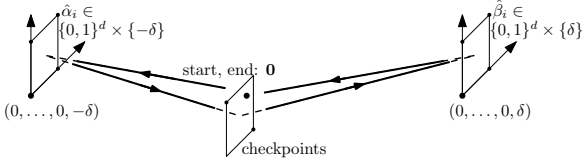


Figure 3: Construction for simplification lower bound.

The idea of the reduction is illustrated in Figure 3. We construct a curve that moves from a starting point through all $\hat{\alpha}_i$, then through d "checkpoints", through all $\hat{\beta}_i$, and finally to an endpoint. The threshold ε for the simplification is chosen such that all points $\hat{\alpha}_i$ have pairwise distance smaller than ε, and similarly for the points $\hat{\beta}_i$. Thus, the intended simplification uses the starting point, one point $\hat{\alpha}_i$, one point $\hat{\beta}_j$, and the endpoint. The checkpoints will have distance at most ε to the edge from $\hat{\alpha}_i$ to $\hat{\beta}_j$ exactly if the two corresponding vectors are orthogonal.

For $1 \le i \le d$, let $q_i \in \mathbb{R}^{d+1}$ be defined as $q_i[i] = -\delta'$, $q_i[d+1] = 0$, and $q_i[\cdot] = 1/4$, otherwise, where δ' will be chosen later depending on the metric. We define a curve $A = \langle a_0, \ldots, a_m\rangle$ with $m = 2n+2+d$ vertices by $a_0 = a_m = (0,\ldots,0)$, $a_i = \hat{\alpha}_i$, for $1 \le i \le n$, $a_{n+i} = q_i$, for $1 \le i \le d$, and $a_{n+d+i} = \hat{\beta}_i$, for $1 \le i \le n$.

We first consider a simplification under the L_∞-metric. We set $\varepsilon = 1$ and $\delta' = 1/2$. By the choice of ε and δ, the simplification needs to include at least a_0, one point $\hat{\alpha}_i$, one point $\hat{\beta}_j$, and a_m. Assume there are orthogonal vectors α_i and β_j. Let $\ell(t)$ be the line segment between $\hat{\alpha}_i$ and $\hat{\beta}_j$ parameterized by t in the $(d+1)$-th coordinate. For the midpoint $\ell(0)$ of the segment we have $\ell(0)[h] = (\hat{\alpha}_i + \hat{\beta}_j)/2 \in \{0, 1/2\}$, for $1 \le h \le d$ (and $\ell(0)[d+1] = 0$). Hence all q_i have distance less than 1 to $\ell(0)$ and are therefore within distance ε to the segment. In contrast, let us assume α_i and β_j are nonorthogonal. In this case there is a coordinate $1 \le h \le d$ such that $\hat{\alpha}_i[h] = \hat{\beta}_j[h] = 1$. It follows that $\ell(t)[h] = 1$ for all $t \in [-\delta, \delta]$, and therefore $d_\infty(\ell(t), q_h) \ge 1 - q_h[h] > 1 = \varepsilon$. Thus, if we choose this segment, q_h has distance larger than ε to the segment. Consequently, if there is no pair of orthogonal vectors, a simplification for distance ε requires at least 5 vertices.

For the L_1-metric we set $\varepsilon = d$ and $\delta' = 3/4d - 1/4$. By the same argument as for L_∞, we get that if there are orthogonal vectors, then they induce a simplification with 4 vertices, since $d_1(q_i, \ell(0)) \le (d-1)/4 + \delta' + 1/2 = d = \varepsilon$. Now again consider the case that all α_i and β_j are nonorthogonal, so there is a coordinate $1 \le h_0 \le d$, such that $\hat{\alpha}_i[h_0] = \hat{\beta}_j[h_0] = 1$. We show that $d_1(\ell(t), q_h) > \varepsilon$ for all $t \in [-\delta, \delta]$. We can restrict our attention to $t \in [-\varepsilon, \varepsilon]$ due to

the $(d+1)$-th coordinate. Now consider a coordinate $h \neq h_0, d+1$. If $\alpha_i[h] = \beta_j[h] = 0$, then $\ell(t)[h] = 0$. Otherwise $\ell(0)[h] \geq 1/2$ and $\ell(t)[h] \geq \ell(0)[h](1-\varepsilon/\delta)$ for $t \in [-\varepsilon, \varepsilon]$. Consequently, for any t we get that $d_1(\ell(t), q_h) \geq (d-1)(1/2(1-\varepsilon/\delta) - 1/4) + 1 - \delta' = [(d-1)/4 + \delta' + 1/2] + 1/2 - (d-1)\varepsilon/\delta = \varepsilon + 1/2 - (d-1)\varepsilon/\delta > \varepsilon$. Thus, there is a simplification using 4 vertices exactly if there is an orthogonal pair.

For the L_2-metric we set $\varepsilon = \sqrt{d}$. Further we fix $\delta' = -1/2 + \sqrt{15d+1}/4$, which implies that $\delta' > 0$ and that $\sqrt{(d-1)/4 + (1/2+\delta')^2} = \varepsilon$. By the choice of δ' orthogonal vectors, we induce points with all q_i having distance at most ε to the segment. Now again consider a pair of nonorthogonal vectors with $\alpha_i[h_0] = \beta_j[h_0] = 1$. It is sufficient then to prove that $d_2(\ell(t), q_h)^2 > \varepsilon^2 = d$ for $t \in [-\varepsilon, \varepsilon]$. Using the same derivation as for L_1, we obtain $d_2(\ell(t), q_h)^2 \geq (1+\delta')^2 + (d-1)(1/4 - \varepsilon/\delta/2)^2$. The first summand is larger than $15/16d + 1/16 + 1/4$ while the second is larger than $(d-1)/16 - (d-1)\varepsilon/\delta/4 > (d-1)/16 - 1/8$. Hence, q_h has a distance larger than ε to the segment.

As a result of this, we can reduce the 2-Orthogonal Vectors problem in dimension d to curve simplification, min-# or min-ε, in dimension $d+1$ for the L_1, L_2 and L_∞ metrics. Theorem 2 follows.

4 Conclusion and Open Problems

We have extended the recent conditional fine-grained hardness results for the Fréchet distance to the case of k curves and to the curve simplification setting, showing that any significant improvement on known methods would result in a major breakthrough in satisfiability algorithms.

We find the curve simplification result particularly intriguing, since in seems to offer a qualitative difference from the previous work: in the curve simplification setting, we have only one input object that needs to be compared to itself. Are there problems of similar flavor where analogous conditional hardness results can be obtained?

Acknowledgements. KB and MK are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.207. WM supported in part by DFG project MU 3501-1.

References

[1] A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proc. 56th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 59–78, 2015.

[2] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014.

[3] P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete Comput. Geom.*, 23(2):273–291, 2000.

[4] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1–2):78–99, 1995.

[5] G. Barequet, D. Z. Chen, O. Daescu, M. T. Goodrich, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica*, 33(2):150–167, 2002.

[6] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 661–670, 2014.

[7] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proc. 56th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 79–97, 2015.

[8] K. Bringmann and W. Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016.

[9] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *Internat. J. Comput. Geom. Appl.*, 21(3):253–282, 2011.

[10] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog - with an application to Alt's conjecture. In *Proc. 25th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1399–1413, 2014.

[11] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 6(01):59–77, 1996.

[12] A. Dumitrescu and G. Rote. On the Fréchet distance of a set of curves. In *Canad. Conf. Comput. Geom. (CCCG)*, pages 162–165, 2004.

[13] H. Imai and M. Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. Elsevier Science, 1988.

[14] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

Time-Space Trade-offs for Triangulating a Simple Polygon*

Boris Aronov† Matias Korman‡ Simon Pratt§ André van Renssen¶‖ Marcel Roeloffzen¶‖

Abstract

An s-workspace algorithm is an algorithm that has read-only access to the values of the input, write-only access to the output and only uses $O(s)$ additional words of space. We give a randomized s-workspace algorithm for triangulating a simple polygon P of n vertices, for any $s \in \Omega(\log n) \cap O(n)$. The algorithm runs in $O(n^2/s)$ expected time. We also extend the approach to compute other similar structures such as the shortest-path map (or tree) of any point $p \in P$, or to partition P using only diagonals of the polygon so that the resulting sub-polygons have $\Theta(s)$ vertices each.

1 Introduction

Triangulation of a simple polygon, often used as a preprocessing step in computer graphics, is performed in a wide range of settings including on embedded systems like the Raspberry Pi or mobile phones. Such systems frequently run read-only filesystems for security reasons and have very limited working memory. An ideal triangulation algorithm for such an environment would allow for a trade-off in performance in time versus working space.

These memory constraints can be modeled by the so-called *s-workspace* model of computation frequently used in the literature (see, for example, [2, 5, 6, 10]). In this model the input data is given in a read-only array or similar structure, and the output we produce must be written to write-only memory.

In our case, the input is a simple polygon P; let $v_1, v_2, \ldots, v_n$ be the vertices of P in clockwise order along the boundary of P. We assume that, given an index i, in constant time we can access the coordinates of the vertex v_i. We also assume that the usual word RAM operations can be performed in constant time (such as, given i, j, k, finding the intersection point of the line passing through vertices v_i and v_j and the horizontal line passing through v_k).

In addition to the read-only data, an s-workspace algorithm can use $O(s)$ variables during its execution, for some parameter s determined by the user. Implicit memory consumption (such as the stack space needed in recursive algorithms) must be taken into account when determining the size of a workspace. We assume that each variable or pointer is stored in a data word of $\Theta(\log n)$ bits. Thus, equivalently, we can say that an s-workspace algorithm uses $O(s \log n)$ bits of storage. In this model, the aim is to design an algorithm whose running time decreases as s grows. Such algorithms are called *time-space trade-off* algorithms [14].

Previous Work

Several variants of this model have been studied (we refer the interested reader to [11] for an overview). In the following we discuss the results related to triangulations. The concept of memory-constrained algorithms was introduced to the computational geometry community by the work of Asano *et al.* [4]. Among other results, they presented an algorithm for triangulating a set of n points in $O(n^2)$ time using $O(1)$ variables. More recently, Korman *et al.* [12] introduced two different time-space trade-off algorithms for the same problem: the first one computes an arbitrary triangulation in $O(n^2/s + n \log n \log s)$ time using $O(s)$ variables. The second is a randomized algorithm that computes the Delaunay triangulation of the given point set in expected $O((n^2/s) \log s + n \log s \log^* s)$ time within the same space bounds.

The first algorithm for triangulating simple polygons was due to Asano *et al.* [2], and runs in $O(n^2)$ time using $O(1)$ variables. Faster algorithms for some particular cases (such as monotone polygons [5, 3]) are also known. To the best of our knowledge, no general time-space trade-off algorithm for simple polygons was previously known. In this paper we introduce a randomized algorithm with expected running time $O(n^2/s)$ that uses $O(s)$ variables to triangulate a simple n-gon, for any $s \in \Omega(\log n) \cap O(n)$. Our approach uses a recent result by Har-Peled [10], which computes the shortest path between two vertices of a simple polygon in expected $O(n^2/s)$ time. Due to lack of space, proofs in this paper are omitted or sketched. Details can be found in the extended version [1].

*Work on this paper by B. A. has been partially supported by NSF Grants CCF-11-17336 and CCF-12-18791. M. K. was supported in part by the ELC project (MEXT KAKENHI No. 24106008). S. P. was supported in part by the Ontario Graduate Scholarship and The Natural Sciences and Engineering Research Council of Canada.

†Tandon School of Engineering, New York University, New York, USA.

‡Tohoku University, Sendai, Japan.

§Cheriton School of Computer Science, University of Waterloo, Canada.

¶National Institute of Informatics (NII), Tokyo, Japan.

‖JST, ERATO, Kawarabayashi Large Graph Project.

2 Preliminaries

We study the problem of computing a triangulation of a simple polygon P in the s-workspace model. A *triangulation* of P is a maximal crossing-free straight-line graph whose vertices are the vertices of P and whose edges lie inside P. Since our workspace is not large enough to store the triangulation explicitly, the goal is to report a triangulation of P in a write-only data structure. After a value is reported, it cannot be accessed or modified.

In preceding similar research [2, 3], the triangulation is reported as a list of edges in no particular order (with no information on neighboring edges or faces). Moreover, it is not clear how to modify these algorithms to obtain such information. Our approach has the advantage that, in addition to the list of edges, we can report adjacency information as well. More details can be found in [1].

Given two points $p, q \in P$, the *geodesic* between them is defined as the shortest path that connects p and q and that stays within P (viewing P as a closed set). The length of that path is called the *geodesic distance*. It is well known that, for any two points of P, their geodesic π always exists and is unique (hence, the geodesic is also often simply referred as the *shortest path* between p and q). Moreover, such a path is a polygonal chain whose vertices (other than p and q) are reflex vertices of P. Thus, we often identify π with the ordered sequence of reflex vertices traversed by the path from p to q. When that sequence is empty (i.e., the shortest path consists of the straight segment pq) we say that p *sees* q (and vice versa).

3 Algorithm

Let π be the geodesic connecting v_1 and $v_{\lfloor n/2 \rfloor}$. From a high-level perspective, the algorithm uses the approach of Har-Peled [10] to compute π, and reports the edges of the shortest path one by one, in order. Our aim is to use this path to subdivide P into smaller subproblems that can be solved recursively.

We start by introducing some definitions that will help in recording which portion of the polygon has already been triangulated. Vertices v_1 and $v_{\lfloor n/2 \rfloor}$ split the boundary of P into two chains. We say that a vertex (other than v_1 and $v_{\lfloor n/2 \rfloor}$) is a *top* vertex if it is in the chain that is traversed when walking along the boundary of P from v_1 to $v_{\lfloor n/2 \rfloor}$ in clockwise fashion or a *bottom* vertex if it lies in the other chain. Note that all vertices, other than v_1 and $v_{\lfloor n/2 \rfloor}$ are either top or bottom vertices. We say that a diagonal c is an *alternating* diagonal if one of its endpoints is a top vertex and the other a bottom vertex (or one of its vertices is either v_1 or $v_{\lfloor n/2 \rfloor}$). Otherwise we say that the diagonal is a *non-alternating* diagonal.

We will use these diagonals to partition P into two parts. Since any two vertices consecutive along the boundary of P can see each other, the partition induced by the "diagonal" connecting them is trivial (i.e., one subpolygon is P and the other is a segment).

Observation 1 *Let c be a diagonal of P such that neither endpoint is v_1 or $v_{\lfloor n/2 \rfloor}$. Vertices v_1 and $v_{\lfloor n/2 \rfloor}$ belong to different components of $P \setminus c$ if and only if c is an alternating diagonal.*

Corollary 1 *Let c be a non-alternating diagonal of P. The component of $P \setminus c$ that contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$ has at most $\lceil n/2 \rceil$ vertices.*

While triangulating the polygon, we maintain an alternating diagonal a_c. Intuitively, the connected component of $P \setminus a_c$ that does not contain $v_{\lfloor n/2 \rfloor}$ has already been triangulated. Since it will prove useful that a_c is not necessarily part of π, we also maintain the property that at least one of the endpoints of a_c will be a vertex of π that has already been computed in the execution of the shortest-path algorithm. Let v_c denote the endpoint of a_c that is on π and that is closest to $v_{\lfloor n/2 \rfloor}$.

With these definitions in place, we can give an intuitive description of our algorithm: we start by setting a_c as the degenerate diagonal from v_1 to v_1. We then use the shortest-path computation approach of Har-Peled. Our aim is to walk along π until we find a new alternating diagonal a_{new}. At this moment we pause the execution of the shortest-path algorithm, triangulate the subpolygons of P that have been created (and contain neither v_1 nor $v_{\lfloor n/2 \rfloor}$) recursively, update a_c to the newly found alternating diagonal, and then resume the execution of the shortest-path algorithm.

Although our approach is intuitively simple, there are several technical difficulties that must be carefully considered. Ideally, the number of diagonals we walked along π is small and can be stored explicitly. But if we do not find an alternating diagonal in just a few steps (indeed, it could even be that there is no alternating diagonal in π), we need to use other diagonals. We also need to make sure that the complexity of each recursive subproblem is reduced by a constant fraction, that we never exceed space bounds, and that no part of the triangulation is reported more than once.

Recall that, at any instant of time, v_c denotes the endpoint of a_c that is in π, and that the subpolygon defined by a_c containing v_1 has already been triangulated. Let $w_0, \ldots, w_k$ be the portion of π up to the next alternating diagonal. That is, path π is of the form $\pi = (v_1, \ldots, v_c = w_0, w_1, \ldots, w_k, \ldots, v_{\lfloor n/2 \rfloor})$ where $w_0w_1, \ldots, w_{k-2}w_{k-1}$ are non-alternating diagonals, and $w_{k-1}w_k$ is an alternating diagonal (or $w_k = v_{\lfloor n/2 \rfloor}$ if no pair of vertices consecutive on π between v_c and $v_{\lfloor n/2 \rfloor}$ forms an alternating diagonal).

Consider the partition of P that these diagonals create, see Figure 1. Let P_1 be the subpolygon induced

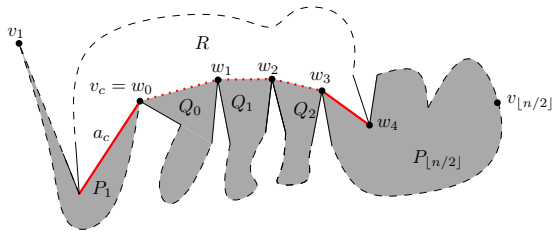


Figure 1: Partitioning P into subpolygons P_1, $P_{\lfloor n/2 \rfloor}$, R, Q_1, …, Q_{k-2}. The two alternating diagonals are marked by thick red lines.

by a_c that does not contain $v_{\lfloor n/2 \rfloor}$. Similarly, let $P_{\lfloor n/2 \rfloor}$ be the subpolygon that is induced by the alternating diagonal $w_{k-1}w_k$ and does not contain v_1. For any $i < k-1$ we define Q_i as the subpolygon induced by the non-alternating diagonal w_iw_{i+1} that contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$. Finally, let R be the remaining component of P. Note that some of these subpolygons may be degenerate and consist only of a line segment (for example, when w_iw_{i+1} is an edge of P).

Lemma 2 *Each of the subpolygons R, Q_1, Q_2, …, Q_{k-2} has at most $\lceil n/2 \rceil + k$ vertices. Moreover, if $w_k = v_{\lfloor n/2 \rfloor}$, then the subpolygon $P_{\lfloor n/2 \rfloor}$ has at most $\lceil n/2 \rceil$ vertices.*

This result allows us to treat the easy case of our algorithm. When k is small (say, a constant number of vertices), we can pause the shortest-path computation algorithm, explicitly store all vertices w_i, recursively triangulate R as well as the subpolygons Q_i (for all $i \leq k-2$), update a_c to the edge $w_{k-1}w_k$, and resume the shortest-path algorithm.

Handling the case where k is large is more involved. Note that we do not know the value of k until we find the next alternating diagonal, but we need not compute it directly. We will be given a parameter τ related to the workspace allowed for our algorithms, and when $k > \tau$, we say that the path is *long*. Initially we set $\tau = s$ but the value of this parameter will change as we descend the recursion tree. We say that the distance between two alternating diagonals is *long* whenever we have computed τ vertices of π besides v_c and no pair of consecutive vertices forms an alternating diagonal. That is, path π is of the form $\pi = (v_1, \ldots, v_c = w_0, w_1, \ldots, w_\tau, \ldots v_{\lfloor n/2 \rfloor})$ and $w_0w_1, \ldots, w_{\tau-1}w_\tau$ are all non-alternating diagonals. In particular, the vertices $w_0, \ldots, w_\tau$ must form a convex chain (see Figure 1). Rather than continue walking along π, we look for a vertex u of P that together with w_τ forms an alternating diagonal. Once we have found this diagonal, we will partition P into $\tau - 2$ subpolygons using the diagonals $a_c, w_0w_1, w_1w_2, \ldots, w_{\tau-1}w_\tau$, and uw_τ similarly to the easy case: P_1 is the part induced by a_c which does not contain $v_{\lfloor n/2 \rfloor}$, $P_{\lfloor n/2 \rfloor}$ is the part induced by uw_τ which does not contain v_1, Q_i is the part induced by the edge w_iw_{i+1} on its boundary, which contains neither v_1 nor $v_{\lfloor n/2 \rfloor}$, and R is the remaining component.

Lemma 3 *We can find a vertex u that together with w_τ forms an alternating diagonal in $O(n)$ time using $O(1)$ space. Moreover, each of the subpolygons R, Q_1, Q_2, …, $Q_{\tau-2}$ has at most $\lceil n/2 \rceil + \tau$ vertices.*

Proof sketch. We use ray shooting to find an edge e outside P_1 which is partially visible to w_τ. Let p_N be one of the endpoints of e. Note that p_N need not be visible to w_τ. However, the triangle formed by w_τ, p_N, and the visible point of e contains one or more reflex vertices. Among those vertices, we know that the vertex r that maximizes the angle $\angle p_N w_\tau r$ must be visible (see Lemma 1 of [6]). As described in Lemma 1 of [6], in order to find such a reflex vertex we need to scan the input polygon at most three times, each time storing a constant amount of information. □

At high level, our algorithm walks from v_1 to $v_{\lfloor n/2 \rfloor}$. We stop after walking τ steps or when we find an alternating diagonal (whichever comes first). This generates several subproblems of smaller complexity that are solved recursively. Once the recursion is done we update a_c (to keep track of the portion of P that has been triangulated), and continue walking along π. The walking process ends when the walk reaches $v_{\lfloor n/2 \rfloor}$. In this case, in addition to triangulating R and the Q_i subpolygons, we must also triangulate $P_{\lfloor n/2 \rfloor}$.

The algorithm on the deeper levels of recursion is almost identical. There are only three minor changes that need to be introduced. First, we compare the size of the polygon to τ rather than s. Recall that τ denotes the amount of space available to the current instance of the algorithm. Thus, if τ is comparable to n (say, $10\tau \geq n$), then the whole polygon fits into memory and can be triangulated in linear time [7]. If τ is significantly smaller, then we continue with the recursive algorithm as usual.

For ease in handling subproblems, at each step we also indicate the vertex that fulfills the role of v_1 (i.e., one of the vertices from which the shortest path must be computed). Recall that we have random access to the vertices of the input. Thus, once we know which vertex takes the role of v_1, we can find the vertex that will satisfy the role of $v_{\lfloor n/2 \rfloor}$ in constant time as well.

In order to avoid exceeding the space bounds, at each level of the recursion we will decrease the value of τ by a factor of $\kappa < 1$.

Theorem 4 *Let P be a simple polygon of n vertices. We can compute a triangulation of P in $O(n^2/s)$ expected time using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$.*

4 Other Applications

The above algorithm introduces a general approach for partitioning P into subpolygons, each of which has at most $O(s)$ vertices. Since our final objective is computing a triangulation, at the bottom level of recursion we use Chazelle's algorithm [7]. However, the same approach can be used for other structures: it suffices to replace the base case of the recursion with an appropriate algorithm. In this section, we mention two examples: computing the shortest-path map and splitting the polygon into pieces of size $\Theta(n)$.

Given a simple polygon P and a point $p \in P$ (which need not be a vertex of P), the *shortest-path tree* of p (denoted by $\mathrm{SPT}(p)$) is the tree formed as the union of all shortest paths from p to vertices of P. ElGindy [8] and later Guibas *et al.* [9] showed how to compute the shortest-path tree in linear time.

The *shortest-path map* of p (denoted by $\mathrm{SPM}(p)$) is the subdivision of P into maximal cells so that points in the same cell have topologically equivalent paths to p. It is well known that $\mathrm{SPM}(p)$ is a finer subdivision than the one induced by $\mathrm{SPT}(p)$. Guibas *et al.* [9, Section 2] showed how to further refine the shortest-path tree so as to obtain the shortest-path map. We refer the interested reader to Lee and Preparata [13] or Guibas *et al.* [9] for more information on shortest-path trees, maps, and their applications.

Theorem 5 *Let P be a simple polygon of n vertices and let p be any point of P (vertex, boundary or interior). We can compute the shortest-path map or shortest-path tree of p in $O(n^2/s)$ expected time using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$.*

Theorem 6 *Let P be a simple polygon of n vertices. For any $s \leq n$, we can partition P with $\Theta(n/s)$ diagonals, so that each subpolygon consists of $\Theta(s)$ vertices, in $O(n^2/s)$ expected time using $O(s)$ variables, for any $s \in \Omega(\log n) \cap O(n)$.*

We note that both Asano *et al.* [2] and Har-Peled [10] already gave methods of partitioning P into subpolygons of roughly the same size. The first one is deterministic, runs in $O(n^2)$ and uses $O(1)$ variables. The one of Har-Peled is a proper trade-off and also runs in $O(n^2/s)$ expected time using $O(s)$ variables. This method introduces additional Steiner points. Our algorithm removes the need for these additional points (since it partitions only by diagonals between visible vertices), while preserving the same running time.

5 Acknowledgments

The authors would like to thank Jean-François Baffier, Man-Kwun Chiu, and Takeshi Tokuyama for valuable discussion in the early stages of the paper.

References

[1] B. Aronov, M. Korman, S. Pratt, A. van Renssen, and M. Roeloffzen. Time-space trade-offs for triangulating a simple polygon. *CoRR*, abs/1509.07669, 2015.

[2] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *CGTA*, 46(8):959–969, 2013.

[3] T. Asano and D. Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *WADS*, pages 61–72, 2013.

[4] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *JoCG*, 2(1):46–68, 2011.

[5] L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

[6] L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing the visibility polygon using few variables. *CGTA*, 47(9):918–926, 2013.

[7] B. Chazelle. Triangulating a simple polygon in linear time. *DCG*, 6:485–524, 1991.

[8] H. A. ElGindy. *Hierarchical Decomposition of Polygons with Applications*. PhD thesis, McGill University, Montreal, Que., Canada, 1985.

[9] L. 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(1–4):209–233, 1987.

[10] S. Har-Peled. Shortest path in a polygon using sublinear space. In *SoCG*, pages 111–125, 2015.

[11] M. Korman. Memory-constrained algorithms. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1–7. Springer Berlin Heidelberg, 2015.

[12] M. Korman, W. Mulzer, M. Roeloffzen, A. v. Renssen, P. Seiferth, and Y. Stein. Time-space trade-offs for triangulations and voronoi diagrams. In *WADS*, pages 482–494, 2015.

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

[14] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.

Separability and Convexity of Probabilistic Point Sets*

Martin Fink † John Hershberger ‡ Nirman Kumar † Subhash Suri †

Abstract

We describe an $O(n^d)$ time algorithm for computing the exact probability that two probabilistic point sets are linearly separable in dimension $d \geq 2$, and prove its hardness via reduction from the k-SUM problem. We also show that d-dimensional separability is computationally equivalent to a $(d+1)$-dimensional convex hull membership problem.

1 Introduction

We consider the problems of linear separability and convex hull membership for probabilistic point sets, where a *probabilistic point* is a tuple (p, π) consisting of a point $p \in \mathbb{R}^d$ and its associated probability of existence π. This abstract representation is a convenient way to model data uncertainty in a number of applications including uncertain databases, sensor networks, data cleansing, scientific computing, and machine learning [4, 5]. We present algorithms and hardness results for computing the exact probability that two such probabilistic sets in $\mathbb{R}^d$ are linearly separable (*separability problem*) or that a point lies inside the convex hull of a probabilistic set (*convex hull membership problem*). Specifically, our results include the following.

1. An $O(n^d)$ time and $O(n)$ space algorithm for computing the probability of separation of two probabilistic point sets with a total of n points in d dimensions, for $d \geq 2$.
2. A reduction from the k-SUM problem to the d-dimensional separability problem, for $k = d+1$, as evidence that our $O(n^2)$ bound for $d = 2$ may be almost tight. We also prove $\#P$-hardness of the problem when $d = \Omega(n)$.
3. A linear-time reduction between the convex hull membership problem in d-space and the separability problem in dimension $(d-1)$.
4. Finally, related problems such as probability of non-empty intersection among n probabilistic halfspaces can also be solved in $O(n^d)$ time. We also show how to extend our result to point sets containing degeneracies.

Related work. The topic of algorithms for probabilistic (uncertain) data is a subject of extensive and ongoing research in many areas of computer science including databases, data mining, machine learning, combinatorial optimization, theory, and computational geometry. Within computational geometry and databases, a number of papers address nearest neighbor searching, minimum spanning trees, Voronoi diagrams, indexing and skyline queries under the probabilistic model of our paper as well as the locational uncertainty model [1, 2, 10, 11, 13, 12]. Our convex hull membership bound improves upon a recent result of [3], both in time complexity and elimination of the non-degeneracy assumption.

2 Separability of Probabilistic Point Sets

2.1 Preliminaries

Let $\mathcal{A}$ and $\mathcal{B}$ be two probabilistic point sets in $\mathbb{R}^d$ with a total of n points. For notational convenience, we denote a generic probabilistic point as p with the implicit understanding that $\pi(p)$ is the probability associated with p and that all the point probabilities are independent. By the independence of probabilities, a subset A occurs as a random sample of $\mathcal{A}$ with probability

$$\mathbf{Pr}[A] = \prod_{p \in A} \pi(p) \cdot \prod_{p \in \mathcal{A} \setminus A} (1 - \pi(p)).$$

We say that the subsets $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{B}$ are linearly separable if there is a hyperplane H containing A and B in opposite (open) halfspaces. (The *open* halfspace separation means that no point of $A \cup B$ lies on H, thus enforcing a strict separation.) Define an indicator function $\sigma(\mathcal{A}, \mathcal{B})$ for linear separability

$$\sigma(A, B) = \begin{cases} 1 & \text{if } A, B \text{ are linearly separable} \\ 0 & \text{otherwise,} \end{cases}$$

with $\sigma(\emptyset, \emptyset) = 1$ to handle the trivial case. Then the *separation probability* of $\mathcal{A}$ and $\mathcal{B}$ is the joint sum over all possible samples:

$$\mathbf{Pr}[\sigma(\mathcal{A}, \mathcal{B})] = \sum_{A \subseteq \mathcal{A}, B \subseteq \mathcal{B}} \mathbf{Pr}[A] \cdot \mathbf{Pr}[B] \cdot \sigma(A, B)$$

This is also the *expectation* of the random variable $\sigma(A, B)$. We are interested in the complexity of computing this quantity *exactly*.

*An expanded version of this work appears in [7].

†University of California, Santa Barbara, CA, USA
‡Mentor Graphics Corp., Wilsonville, OR, USA

2.2 Reduction to Anchored Separability

There are $O(n^d)$ combinatorially distinct separating hyperplanes induced by $\mathcal{A} \cup \mathcal{B}$, so a natural idea is to decompose the sum into probabilities over these planes. However, many different hyperplanes may be separating for the same sample pair, so we must avoid over-counting by assigning each pair to a unique *canonical* hyperplane.[1] Our main insight is the following: if we introduce an extra point z into the input, then the canonical hyperplane can be defined uniquely (and computed efficiently) with respect to z. We call this additional point z the *anchor point*.

We initially assume that the input points are in general position, and choose z *above* (in the dth coordinate) all the input points and in general position with respect to $\mathcal{A} \cup \mathcal{B}$. The non-degeneracy assumption can be eliminated, as briefly explained in Section 5. We assign $\pi(z) = 1$ so that the anchor is always included in the sample.

If $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{B}$ are two random samples and H a hyperplane separating them, then z lies either (i) on the same side as A, (ii) on the same side as B, or (iii) on the hyperplane H. The following lemma shows that case (iii) precisely counts the double-counting between cases (i) and (ii).

Lemma 1 *There exist separating hyperplanes H_1, H_2 with z lying on the same side of H_1 as A but on the same side of H_2 as B if and only if there is another hyperplane H that passes through z and separates A from B.*

Let $\mathcal{P} + z$ be the shorthand for the probabilistic point set $\mathcal{P} \cup \{z\}$, with $\pi(z) = 1$. Let $\mathbf{Pr}\big[\sigma(z, \mathcal{A}, \mathcal{B})\big]$ denote the probability that sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable by a hyperplane passing through z. By the preceding lemma, we have the following.

Lemma 2 *Given two probabilistic point sets $\mathcal{A}$ and $\mathcal{B}$, we have the following equality:*

$$\begin{aligned}\mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B})\big] &= \mathbf{Pr}\big[\sigma(\mathcal{A}+z, \mathcal{B})\big] + \mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B}+z)\big] \\ &- \mathbf{Pr}\big[\sigma(z, \mathcal{A}, \mathcal{B})\big].\end{aligned}$$

Computing $\mathbf{Pr}\big[\sigma(\mathcal{A}+z, \mathcal{B})\big]$ and $\mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B}+z)\big]$ requires solving two instances of *anchored separability*, once with z included in $\mathcal{A}$ and once in $\mathcal{B}$, and this is the problem we solve in the following subsection. The calculation of the remaining term $\mathbf{Pr}\big[\sigma(z, \mathcal{A}, \mathcal{B})\big]$ can be reduced to an instance of separability in dimension $d-1$, as shown below.

Consider any sample $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{B}$. We centrally project all these points onto the hyperplane $x_d = 0$ from the anchor point z: the image of a point $p \in \mathbb{R}^d$ is the point $p' \in \mathbb{R}^{d-1}$ at which the line connecting z to p intersects the hyperplane $x_d = 0$. All points of $\mathcal{A} \cup \mathcal{B}$ have a well-defined projection because z lies above all of them.

Lemma 3 *Let $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{B}$ be two sample sets, and let A', B' be their projections onto $x_d = 0$ with respect to z. Then A and B are separable by a hyperplane passing through z if and only if A' and B' are linearly separable in $x_d = 0$.*

3 Computing Anchored Separability

We now describe our main technical result, namely, how to compute the probability of anchored separability $\mathbf{Pr}\big[\sigma(\mathcal{A}+z, \mathcal{B})\big]$. Given a hyperplane H, we can easily compute the probability that $\mathcal{A}+z$ lies in H^+ and $\mathcal{B}$ lies in H^-. The separation probabilities for different hyperplanes, however, are not independent, and so our algorithm "assigns" each separable sample to a unique hyperplane, which geometrically is the hyperplane that separates $A+z$ from B and lies at *maximum distance* from the anchor z. We introduce the concept of a *shadow cone* to formalize these canonical hyperplanes (see Fig. 1).

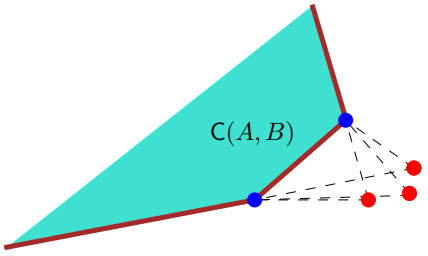


Figure 1: A shadow cone in two dimensions.

Given two points $u, v \in \mathbb{R}^d$, let $\mathit{shadow}(u, v) = \{\lambda v + (1-\lambda)u \mid \lambda \geq 1\}$ be the ray originating at v and directed along the line uv away from u. Given two sets of points A and B, with $A \cap B = \emptyset$, we define their *shadow cone* $\mathsf{C}(A, B)$ as the union of $\mathit{shadow}(u, v)$ for all $u \in CH(A)$ and $v \in CH(B)$, where $CH()$ denotes the convex hull.

$\mathsf{C}(A, B)$ is a (possibly unbounded) convex polytope, each of whose faces is defined by a subset of (at most d) points in $A \cup B$, and the defining set always includes at least one point of B. The following lemma states the important connection between the shadow cone and hyperplane separability.

Lemma 4 *$A+z$ and B can be separated by a hyperplane if and only if $z \notin \mathsf{C}(A, B)$.*

3.1 Canonical Separating Hyperplanes

Since $\mathsf{C}(A, B)$ is a convex set, there is a *unique* nearest point $p = \mathrm{np}(z, \mathsf{C}(A, B))$ on the boundary of $\mathsf{C}(A, B)$

[1] Dualizing the points to hyperplanes can simplify the enumeration of separating planes for the summation but does not address the over-counting problem.

with minimum distance to z. We define our *canonical hyperplane* $H(z, A, B)$ as the one that passes through p and is orthogonal to the vector $p - z$. The following lemma states the definition of canonical separators.

Lemma 5 *Let C be a d-dimensional convex polyhedron, z a point not contained in C, and p the point of C at minimum distance from z. If p lies in the relative interior of the face F of C, then the hyperplane H through p that is orthogonal to $p - z$ contains F. This hyperplane contains C in one of its closed halfspaces, and is the hyperplane farthest from z with this property.*

We turn the separation question around and instead of asking "which hyperplane separates a particular sample pair A, B," we ask "for which pairs of samples A, B is H a canonical separator?" The latter formulation allows us to compute the separation probability $\mathbf{Pr}\big[\sigma(\mathcal{A}+z, \mathcal{B})\big]$ by considering at most $O(n^d)$ possible hyperplanes.

3.2 The Algorithm

Our algorithm enumerates all subsets $I \subseteq \mathcal{A}$ and $J \subseteq \mathcal{B}$, with $|I \cup J| \leq d$ and $|J| \geq 1$, and assigns to the hyperplane $H(z, I, J)$ the separation probability of *all those samples $A \cup B$ that are separable and for which $H(z, I, J)$ is the canonical separator $H(z, A, B)$*. Let $\mathbf{Pr}\big[H(z, I, J)\big]$ denote the probability that the points defining the hyperplane $H(z, I, J)$ are in the sample and none of the remaining points of $\mathcal{A} \cup \mathcal{B}$ lies on its *incorrect side*. Then, it's easy to check that

$$\begin{aligned}\mathbf{Pr}\big[H(z,I,J)\big] &= \prod_{u \in I \cup J} \pi(u) \times \prod_{u \in \mathcal{A} \cap H^-} (1 - \pi(u)) \\ &\times \prod_{u \in \mathcal{B} \cap H^+} (1 - \pi(u)).\end{aligned}$$

The pseudo-code below describes our algorithm.

Algorithm AnchoredSep:

Input: The point sets $\mathcal{A} + z$ and $\mathcal{B}$
Output: Their separation probability
$\alpha = \mathbf{Pr}\big[\sigma(\mathcal{A} + z, \mathcal{B})\big]$
$\alpha = \prod_{u \in \mathcal{B}}(1 - \pi(u))$;
forall the $I \subseteq \mathcal{A}, J \subseteq \mathcal{B}$ *where* $|I \cup J| \leq d, J \neq \emptyset$ **do**
  let $p = \mathrm{np}(z, \mathsf{C}(I, J))$;
  if *p lies in the relative interior of* $\mathsf{C}(I, J)$ **then**
    $\alpha = \alpha + \mathbf{Pr}\big[H(z, I, J)\big]$;
  end
end
return α;

Theorem 6 AnchoredSep *correctly computes the probability* $\mathbf{Pr}\big[\sigma(\mathcal{A} + z, \mathcal{B})\big]$.

A naïve implementation of **AnchoredSep** runs in $O(n^{d+1})$ time and $O(n)$ space, but it can be improved to $O(n^d)$ time using duality and topological sweep.

Theorem 7 *Let $\mathcal{A}, \mathcal{B} \subseteq \mathbb{R}^d$ be two probabilistic sets of n points in general position, for $d \geq 2$. We can compute their probability of hyperplane separation $\mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B})\big]$ in $O(n^d)$ worst-case time.*

4 Lower Bounds

We now argue that the separability problem is at least as hard as the k-SUM problem for $k = d + 1$, for any fixed d. We also show that the problem is $\#P$-hard when $d = \Omega(n)$.

The k-SUM problem is a generalization of 3-SUM, which is a classical hard problem in computational geometry [8, 9]. We use the following variant: Given k sets containing a total of n real numbers, grouped into a single set Q and $k - 1$ sets $R_1, R_2, \ldots, R_{k-1}$, determine whether there exist $k - 1$ elements $r_i \in R_i$, one per set R_i, and an element $q \in Q$ such that $\sum_{i=1}^{k-1} r_i = q$. We have the following result.

Theorem 8 *The d-dimensional hyperplane separability problem is at least as hard as $(d + 1)$-SUM.*

The problem is $\#P$-hard for $d = \Omega(n)$.

Lemma 9 *Computing $\mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B})\big]$ is $\#P$-hard if the dimension d is not a constant.*

Proof. We reduce the $\#P$-hard problem of counting independent sets in a graph [14] to the separability problem. Consider an undirected graph $G = (V, E)$ on the vertex set $\{1, 2, \ldots, n\}$. For each i, we construct an n-dimensional point $a_i = (0, \ldots, 1, \ldots, 0)$, namely, the unit vector along the ith axis. The collection of points $\{a_1, \ldots, a_i, \ldots, a_n\}$, each with associated probability $\pi_i = 1/2$, is our point set $\mathcal{A}$. Next, for each edge $e = (i, j) \in E$, we construct a point b_{ij} at the midpoint of the line segment connecting a_i and a_j. The set of points b_{ij}, each with associated probability 1, is the set $\mathcal{B}$. It is easy to see that there is a one-to-one correspondence between separable subsets of $\mathcal{A} \cup \mathcal{B}$ and the independent sets of G. Each separable sample occurs precisely with probability $(1/2)^n$, and therefore we can count the number of independent sets using the separation probability $\mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B})\big]$. □

5 Handling Input Degeneracies

We deal with degenerate inputs through a problem-specific symbolic perturbation within the framework

of Simulation of Simplicity [6]. We convert degenerate non-separable samples into non-degenerate samples that are still non-separable. We first choose the anchor z above all points in $\mathcal{P} = \mathcal{A} \cup \mathcal{B}$ and outside the affine span of every d-tuple of $\mathcal{P}$. For each $a \in \mathcal{A}$, we define a perturbed point $a' = a + \epsilon \cdot (a - z)$, and for each $b \in \mathcal{B}$, define $b' = b + \epsilon \cdot (z - b)$, where $\epsilon > 0$ is infinitesimally small. Let $\mathcal{A}', \mathcal{B}'$ be the sets of perturbed points corresponding to $\mathcal{A}$ and $\mathcal{B}$. We prove that $A + z$ and B are strictly separable by a hyperplane if and only if $A' + z$ and B' are. Furthermore, if some hyperplane H with $z \notin H$ is a non-strict separator of $A' + z$ and B' for some ϵ, then H is a strict separator for any $\epsilon_0 < \epsilon$.

6 Convexity and Related Problems

Given a probabilistic set of points $\mathcal{P}$, the convex hull membership probability of a query point z is the probability that z lies in the convex hull of $\mathcal{P}$. We write this as $\mathbf{Pr}\big[z \in CH(\mathcal{P})\big] = \sum_{P \subseteq \mathcal{P},\ z \in CH(P)} \mathbf{Pr}\big[P\big]$. Without loss of generality, assume that the query point is $z = (0, 0, \ldots, 1)$, and define the *central projection* of $p \in \mathcal{P}$ as the point p' at which the line pz meets the plane $x_d = 0$. Let the set $\mathcal{A}$ (resp. $\mathcal{B}$) be the central projections of all those points in $\mathcal{P}$ with $x_d > 1$ (resp. with $x_d < 1$), where each point inherits the associated probability of its corresponding point in $\mathcal{P}$. The sets $\mathcal{A}$ and $\mathcal{B}$ are $(d-1)$-dimensional probabilistic points, with $|\mathcal{A}| + |\mathcal{B}| = n$. We show the following equality

$$\mathbf{Pr}\big[z \in CH(\mathcal{P})\big] = 1 - \mathbf{Pr}\big[\sigma(\mathcal{A}, \mathcal{B})\big],$$

which proves that d-dimensional convex hull membership can be computed in the same time bound as the $(d-1)$-dimensional separability. Similarly, the probability that n probabilistic halfspaces have non-empty intersection can be computed in the same time bound as d-dimensional separability.

7 Concluding Remarks

We considered the problem of hyperplane separability for probabilistic point sets. Our main result is that, given two sets of n probabilistic points in $\mathbb{R}^d$, we can compute in $O(n^d)$ time the exact probability that their random samples are linearly separable. The same technique and result lead to similar bounds for several other problems, including the probability that a query point lies inside the convex hull of n probabilistic points, or the probability that n probabilistic halfspaces have non-empty intersection. We also proved that the d-dimensional separability problem is at least as hard as the $(d+1)$-SUM problem [8, 9], which implies that our $O(n^2)$ algorithms for 2-dimensional separability or 3-dimensional convex hull membership are nearly optimal.

References

[1] P. K. Agarwal, S. W. Cheng, and K. Yi. Range searching on uncertain data. *ACM Trans. on Algorithms*, 8(4):43:1–43:17, 2012.

[2] P. K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang. Nearest-neighbor searching under uncertainty. In *PODS*, pages 225–236, 2012.

[3] P. K. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang. Convex hulls under uncertainty. In *Proc. 22nd ESA*, pages 37–48, 2014.

[4] C. C. Aggarwal. *Managing and Mining Uncertain Data*. Springer, 2009.

[5] C. C. Aggarwal and P. S. Yu. A survey of uncertain data algorithms and applications. *IEEE TKDE.*, 21(5):609–623, 2009.

[6] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. on Graphics*, 9(1):66–104, 1990.

[7] M. Fink, J. Hershberger, N. Kumar, and S. Suri. Hyperplane separability and convexity of probabilistic point sets. In *Proc. 32nd SoCG (to appear)*, 2016.

[8] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *CGTA*, 5(3):165–185, 1995.

[9] A. Gronlund and S. Pettie. Threesomes, degenerates, and love triangles. In *Proc. 55th FOCS*, pages 621–630, 2014.

[10] P. Kamousi, T. M. Chan, and S. Suri. Stochastic minimum spanning trees in Euclidean spaces. In *Proc. 27th SoCG*, pages 65–74, 2011.

[11] H. P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *Advances in Databases: Concepts, Systems and Applications*, pages 337–348. 2007.

[12] S. Suri and K. Verbeek. On the most likely Voronoi diagram and nearest neighbor searching. In *Proc. 25th ISAAC*, pages 338–350, 2014.

[13] S. Suri, K. Verbeek, and H. Yıldız. On the most likely convex hull of uncertain points. In *Proc. 21st ESA*, pages 791–802, 2013.

[14] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

Finding Plurality Points in $\mathbb{R}^{d*}$

Mark de Berg[†] Joachim Gudmundsson[‡] Mehran Mehr[§]

Abstract

Let V be a set of n points in $\mathbb{R}^d$, which we call voters, where d is a fixed constant. A point $p \in \mathbb{R}^d$ is preferred over another point $p' \in \mathbb{R}^d$ by a voter $v \in V$ if $\text{dist}(v,p) < \text{dist}(v,p')$. A point p is called a plurality point if it is preferred by at least as many voters as any other point p'.

We present an algorithm that decides in $O(n \log n)$ time whether V admits a plurality point in the L_2 norm and, if so, finds the (unique) plurality point.

1 Introduction

We study computational problems concerning plurality points, a concept arising in social choice and voting theory, defined as follows. Let V be a set of n *voters* and let $\mathcal{C}$ be a space of possible choices. Each voter $v \in V$ has a utility function indicating how much v likes a certain choice. Thus the utility function of v determines for any two choices from $\mathcal{C}$ which one is preferred by v or whether both choices are equally preferable. A (weak) plurality point is now defined as a choice $p \in \mathcal{C}$ such that no alternative $p' \in C$ is preferred by more voters.

When there are different issues on which the voters can decide, then the space $\mathcal{C}$ becomes a multi-dimensional space. This has led to the study of plurality points in the setting where $\mathcal{C} = \mathbb{R}^d$ and each voter has an ideal choice which is a point in $\mathbb{R}^d$. To simplify the presentation, from now on we will not distinguish the voters from their ideal choice and so we view each voter $v \in V$ as being a point in $\mathbb{R}^d$, the so-called *spatial model* in voting theory [10]. Thus the utility of a point $p \in \mathbb{R}^d$ for a voter v is inversely proportional to $\text{dist}(v,p)$, the distance from v to p under a given distance function, and v prefers a point p over a point p' if $\text{dist}(v,p) < \text{dist}(v,p')$. Now a point $p \in \mathbb{R}^d$ is a plurality point if for any point $p' \in \mathbb{R}^d$ we have $|\{v \in V : \text{dist}(v,p) < \text{dist}(v,p')\}| \geqslant |\{v \in V : \text{dist}(v,p') < \text{dist}(v,p)\}|$.

Plurality points and related concepts were already studied in the 1970s in voting theory [4, 6, 7, 10, 12]. McKelvey and Wendell [10] define three different notions of plurality points—majority Condorcet, plurality Condorcet, and majority core—and for each notion they define a weak and a strong variant. Under certain assumptions on the utility functions, which are satisfied for the L_2 norm, the three notions are equivalent. Thus for the L_2 norm we only have two variants: weak plurality points (which should be at least as popular as any alternative) and strong plurality points (which should be strictly more popular than any alternative). We focus on weak plurality points, since they are more challenging from an algorithmic point of view. From now on, whenever we speak of plurality points we refer to weak plurality points.

Plurality points represent a stable choice with respect to the opinions of the voters. One can also look at the concept from the viewpoint of competitive facility location. Here one player wants to place a facility in the space $\mathcal{C}$ such that she always wins at least as many clients (voters) as her competitor, no matter where the competitor places his facility. Competitive facility location problems have been studied widely in a discrete setting, where the clients and the possible locations for the facilities are nodes in a network; see the survey by Kress and Pesch [8]. Competitive facility location has also been studied in a geometric, continuous setting under the name Voronoi games [1, 3]. Here one is given a region R in $\mathbb{R}^2$, say the unit square, and the goal is to win the maximum area within R. In other words, the set V of voters is no longer finite, but we have $V = \mathcal{C} = R$. The plurality-point problem in a geometric space lies in between the network setting and the fully continuous setting: the space $\mathcal{C}$ of choices is $\mathbb{R}^d$, but the set V of voters is finite.

When the L_2 norm defines the distance between voters and potential plurality points, then plurality points can be defined in terms of Tukey depth [11]. The *Tukey depth* of a point $p \in \mathbb{R}^d$ with respect to a given set V of n points is defined as the minimum number of points from V lying in any closed halfspace containing p. A point of maximum Tukey depth is called a *Tukey median*. It is known that for any set V, the depth of the Tukey median is at least $\lceil n/(d+1) \rceil$ and at most $\lceil n/2 \rceil$. Wu *et al.* [13] showed that a point $p \in \mathbb{R}^d$ is a plurality point in the L_2 norm if

*MdB and MM are supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003 and 022.005025, respectively. JG supported under Australian Research Council's Discovery Projects funding scheme (project number DP150101134).

[†]Department of Computer Science, TU Eindhoven, the Netherlands, mdberg@win.tue.nl

[‡]School of IT, University of Sydney, Australia, joachim.gudmundsson@sydney.edu.au

[§]Department of Computer Science, TU Eindhoven, the Netherlands, mmehr@tue.nl

and only if any open halfspace with p on its boundary contains at most $n/2$ voters. This is equivalent to saying that the Tukey depth of p is $\lceil n/2 \rceil$. They used this observation to present an algorithm that decides in $O(n^{d-1} \log n)$ time if a given set V of n voters in $\mathbb{R}^d$ admits a plurality point with respect to the L_2 norm and, if so, finds such a point. A slightly better result can be obtained using a randomized algorithm by Chan [2], which computes a Tukey median (together with its depth) in $O(n \log n + n^{d-1})$ time.

Our results. Currently the fastest algorithm for deciding whether a plurality point exists runs in $O(n \log n + n^{d-1})$ randomized time and actually computes a Tukey median. However, in the case of plurality points we are only interested in the Tukey median if its depth is the maximum possible, namely $\lceil n/2 \rceil$. Wu *et al.* [13] exploited this to obtain a deterministic algorithm, but their running time is $O(n^{d-1} \log n)$. This raises the question: can we decide whether a plurality point exists faster than by computing the depth of the Tukey median? We show that this is indeed possible: we present a deterministic algorithm that decides if a plurality point exists (and, if so, computes one) in $O(n \log n)$ time.

2 Plurality points in the L_2 norm

Let V be a set of n voters in $\mathbb{R}^d$. In this section we show how to compute a plurality point for V with respect to the L_2 norm in $O(n \log n)$ time, if it exists. We start by proving several properties of the plurality point in higher dimensions, which generalize similar properties that Lin *et al.* [9] proved in $\mathbb{R}^2$. These properties imply that if a plurality point exists, it is unique (unless all points are collinear). Our algorithm then consists of two steps: first it computes a single candidate point $p \in \mathbb{R}^d$, and then it decides if p is a plurality point.

2.1 Properties of plurality points in the L_2 norm

As remarked in the introduction, plurality points can be characterized as follows.

Fact 1 (Wu *et al.* [13]) *A point p is a plurality point for a set V of n voters in $\mathbb{R}^d$ with respect to the L_2 norm if and only if every open halfspace with p on its boundary contains at most $n/2$ voters.*

Verifying the condition in Fact 1 directly is not efficient. Hence, we will prove alternative conditions for a point p to be a plurality point in $\mathbb{R}^d$, which generalize the conditions Lin *et al.* [9] stated for the planar case. First, we define some concepts introduced by Lin *et al.*

Let V be a set of n voters in $\mathbb{R}^d$, and consider a point $p \in \mathbb{R}^d$. Let $L(p)$ be the set of all lines passing through p and at least one voter $v \neq p$. The point p partitions each line $\ell \in L(p)$ into two opposite rays, which we denote by $\rho(\ell)$ and $\overline{\rho}(\ell)$. (The point p itself is not part of these rays.) We say that a line $\ell \in L(p)$ is *balanced* if $|\rho(\ell) \cap V| = |\overline{\rho}(\ell) \cap V|$. If n is odd, then p is a plurality point if and only if every line $\ell \in L(p)$ is balanced (which implies that we must have $p \in V$). When n is even the situation is more complicated. Let $R(p)$ be the set of all rays $\rho(\ell)$ and $\overline{\rho}(\ell)$. Label each ray in $R(p)$ with an integer, which is the number of voters on the ray minus the number of voters from V on the opposite ray. Thus, a line ℓ is balanced if and only if its rays $\rho(\ell)$ and $\overline{\rho}(\ell)$ have label zero. Let $L^*(p)$ be the set of all unbalanced lines in $L(p)$ and let $R^*(p)$ be the corresponding set of rays. We now define the so-called alternating property, as introduced by Lin *et al.* [9]. This property is restricted to the 2-dimensional setting, where we can order the rays in $R^*(p)$ around p. In this setting, the point p is said to have the *alternating property* if the following holds: the circular sequence of labels of the rays in $R^*(p)$, which we obtain when we visit the rays in $R^*(p)$ in clockwise order around p, alternates between labels $+1$ and -1. Note that if p has the alternating property then the number of unbalanced lines must be odd.

Theorem 2 *Let V be a set of n voters in $\mathbb{R}^d$, with $d \geqslant 1$, and let p be an arbitrary point.*

a. *If n is odd, p is a plurality point if and only if $p \in V$ and every line in $L(p)$ is balanced.*

b. *If n is even and $p \notin V$, then p is a plurality point if and only if every line in $L(p)$ is balanced.*

c. *If n is even and $p \in V$, then p is a plurality point if and only if all unbalanced lines in $L(p)$ are contained in a single 2-dimensional flat f and p has the alternating property for the set $V \cap f$.*

For $d = 1$ the theorem is trivial, and for $d = 2$—the condition in case c then simply states that p has the alternating property—the theorem was proved by Lin *et al.* [9] Our contribution is the extension to higher dimensions. Before proving Theorem 2, we need the following lemma regarding the robustness of plurality points to dimension reduction.

Lemma 3 *Let p be a plurality point for a set V in $\mathbb{R}^d$, with $d \geqslant 1$, and let f be any lower-dimensional flat containing p. Then p is a plurality point for $V \cap f$.*

Proof. We prove the statement by induction on d. For $d = 1$ the lemma is trivially true, so now consider the case $d > 1$. We consider two cases.

The first case is that f is a hyperplane, that is, $\dim(f) = d - 1$. Let f^+ and f^- denote the open halfspaces bounded by f, and assume without loss of generality that $|f^+ \cap V| \geqslant |f^- \cap V|$. Suppose for a

contradiction that p is not a plurality point for $f \cap V$. Then there must be a $(d-2)$-flat $g \subset f$ containing p such that, within the $(d-1)$-dimensional space f, the number of voters lying strictly to one side of g is greater than $|f \cap V|/2$. Let $g^+ \subset f$ denote the part of f lying to this side of g. Now imagine rotating f around g by an infinitesimal amount. Let $\hat{f}$ denote the rotated hyperplane. Then all voters in $f^+ \cap V$ end up in $\hat{f}^+$. Moreover, we can choose the direction of the rotation such that the voters in $g^+ \cap V$ end up in $\hat{f}^+$. But then $|\hat{f}^+ \cap V| = |f^+ \cap V| + |g^+ \cap V| > |f^+ \cap V| + |f \cap V|/2 \geqslant n/2$, which contradicts the assumption that p is a plurality point.

The second case is that $\dim(f) < d-1$. Let h be a hyperplane that contains f. From the first case we know that p must be a plurality point for $h \cap V$. Hence, we can apply our induction hypothesis to conclude that p must be a plurality point for $f \cap V$. □

Now we are ready to prove Theorem 2.

Proof. [Proof of Theorem 2] Since the case $d = 2$ was already proved by Lin *et al.* [9], and the case $d = 1$ is trivial, we assume $d \geqslant 3$. Below we prove part c. The proof of parts a and b is given in the full version.

(c,⇐). Assume n is even and let p be a point such that all unbalanced lines in $L(p)$ are contained in a single 2-dimensional flat f and p has the alternating property for the set $V \cap f$. Consider an arbitrary open halfspace h^+ whose bounding hyperplane h contains p, and let h^- be the opposite open halfspace. If h contains f then all unbalanced lines lie in h and so $|h^+ \cap V| = |h^- \cap V|$, which implies $|h^+ \cap V| \leqslant n/2$. If h does not contain f, we can argue as follows. Let $\ell := h \cap f$. Since the theorem is true for $d = 2$ and we have the alternating property on f, we know that p is a plurality point on f. Hence, the number of voters on f on either side of ℓ is at most $|f \cap V|/2$. But then we have $|h^+ \cap V| \leqslant n/2$, because all voters not in f lie on balanced lines. We conclude that for any open halfspace h^+ we have $|h^+ \cap V| \leqslant n/2$, and so p is a plurality point.

(c,⇒). Assume n is even and let p be a plurality point. We first argue that all unbalanced lines must lie on a single 2-flat. Assume for a contradiction that there are three unbalanced lines that do not lie on a common 2-flat. Let g be the 3-flat spanned by these lines, and let $L^*(g) \subset L^*(p)$ be the set of all unbalanced lines contained in g. Let $f_1 \subset g$ be a 2-flat not containing p and not parallel to any of the lines in $L^*(g)$. Each of the lines in $L^*(g)$ intersects f_1 in a single point, and these intersection points are not all collinear. According to the Sylvester-Gallai Theorem [5] this implies there is an ordinary line in f_1, that is, a line containing exactly two of the intersection points. Thus we have an ordinary 2-flat in g, that is, a flat f_2 containing exactly two lines from $L^*(p)$. This implies that $f_2 \cap V$ does not have the alternating property, and since we know by the result of Lin *et al.* that the theorem holds when $d = 2$ this implies that p is not a plurality point in f_2. However, this contradicts Lemma 3.

We just argued that all unbalanced lines must lie on a single 2-flat f. By Lemma 3 the point p is a plurality point on f. Since the theorem holds for $d = 2$, we conclude that $f \cap V$ has the alternating property.

□

2.2 Finding plurality points in the L_2 norm

We now turn our attention to finding a plurality point. Our algorithm needs a subroutine for finding a *median hyperplane* h for V, which is a hyperplane such that $|h^+ \cap V| < n/2$ and $|h^- \cap V| < n/2$, where h^+ and h^- denote the two open halfspaces bounded by h. The following lemma is easy to prove.

Lemma 4 *Let $v \in V$ be a voter that lies on a hyperplane h_0 such that all voters either lie on h_0 or in h_0^+. Then we can find a median hyperplane h containing v in $O(n)$ time.*

For $d \geqslant 2$ the plurality point is unique, if it exists (the proof is given in the full version). The algorithm below either reports a single candidate point p—we show later how to test if the candidate is actually a plurality point or not—or it returns $\emptyset$ to indicate that it already discovered that a plurality point does not exist. When called with a set V of n collinear voters, the algorithm will return the set of all plurality points; if n is even the set is a segment connecting the two median voters, if n is odd the set is a degenerate segment consisting of the (in this case unique) median voter. We call this segment the *median segment*.

FindCandidates(V)

1. If all voters in V are collinear, then return the median segment of V.
2. Otherwise, proceed as follow.
   (a) Let $v_0 \in V$ be a voter with minimum x_d-coordinate. Find a median hyperplane h_0 containing v_0 using Lemma 4, and let $cand_0 :=$ FindCandidates($h_0 \cap V$).
   (b) If $cand_0$ is a single point or $cand_0 = \emptyset$ then return $cand_0$.
   (c) If $cand_0$ is a (non-degenerate) segment then let $v_1 \in V$ be a voter whose distance to h_0 is maximized. Find a median hyperplane h_1 containing v_1 using Lemma 4, and let $cand_1 :=$ FindCandidates($h_1 \cap V$). Return $cand_0 \cap cand_1$.

Lemma 5 *Algorithm* FindCandidates(V) *returns in $O(n)$ time a set* cand *of candidate plurality points such that (i) if all voters in V are collinear then* cand *is the set of all plurality points of V; (ii) otherwise* cand *contains at most one point, and no other point can be a plurality point of V.*

Proof. If all voters in V are collinear then the algorithm returns the correct result in Step 1, so assume not all voters are collinear. Consider the median hyperplane computed in Step 2a. Since $|h_0^+ \cap V| < n/2$ and $|h_0^- \cap V| < n/2$, for any point $p \notin h$ there is an open halfspace containing p and bounded by a hyperplane parallel to h_0 that contains more than $n/2$ voters. Hence, by Fact 1 any plurality point for V must lie on h_0. By Lemma 3, if a plurality point exists for V it must also be a plurality point for $h_0 \cap V$. By induction we can assume that FindCandidates($h_0 \cap V$) is correct. Hence, the result of the algorithm is correct when $cand_0$ is a single point or $cand_0 = \emptyset$. Note that when $cand_0$ is a (non-degenerate) segment—this only happens when all voters in $h_0 \cap V$ are collinear—we must have $V \neq h_0 \cap V$, otherwise V would be collinear and we would be done after Step 1. Hence, $v_1 \notin h_0$. By the same reasoning as above the median hyperplane h_1 must contain the plurality point of V (if it exist). But then the plurality point must lie in $cand_0 \cap cand_1$, and since $v_1 \notin h_0$ we know that $cand_0 \cap cand_1$ is either a single point or it is empty. This proves the correctness.

To prove the time bound, we note that we only have two recursive calls when the first recursive call reports a non-degenerate candidate segment. This only happens when all voters in $h_0 \cap V$ are collinear, which implies the recursive call just needs to compute a median segment in $O(n)$ time—it does not make further recursive calls. Thus we can imagine adding this time to the original call, so that we never make more than one recursive call. Since the recursion depth is at most d, and each call needs $O(n)$ time, the bound follows. □

Our algorithm to find a plurality point first calls FindCandidates(V). If all points in V are collinear we are done—FindCandidates(V) then reports the correct answer. Otherwise we either get a single candidate point p, or we already know that a plurality point does not exist. It remains to test if a candidate point p is a plurality point or not.

To this end we have to check the conditions of Theorem 2, which can easily be done in $O(n \log n)$ time.

Lemma 6 *Given a set V of n voters in $\mathbb{R}^d$ and a candidate point p, we can test in $O(n \log n)$ time if p is a plurality point in the L_2 norm.*

We obtain the following theorem.

Theorem 7 *Let V be a set of n voters in $\mathbb{R}^d$, where $d \geqslant 2$ is a fixed constant. Then we can find in $O(n \log n)$ time the plurality point for V in the L_2 norm, if it exists, and this time bound is optimal.*

3 Conclusion

Most point sets do not admit a plurality point in the L_2 norm. Hence, in the full version of the paper we also study several other problems concerning plurality points: we give fast algorithms to find the smallest subset $W \subset V$ such that $V \setminus W$ admits a plurality point, we show how to compute a so-called plurality ball in the plane, and we study plurality points in the L_1 norm.

References

[1] H.-K. Ahn, S.-W. Cheng, O. Cheong, M. Golin, and R. van Oostrum. Competitive facility location: the Voronoi game. *Theor. Comput. Sci.*, 310(1–3):457–467, 2004.

[2] T. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proc. 15th ACM-SIAM Symp. Discr. Alg. (SODA)*, pages 430–436, 2004.

[3] O. Cheong, S. Har-Peled, N. Linial, and J. Matousek. The one-round Voronoi game. *Discr. Comput. Geom.* 31(1):125–138, 2004.

[4] H. A. Eiselt and G. Laporte. Sequential location problems. *Europ. J. Op. Res.* 96(2):217–231, 1997.

[5] T. Gallai. Solution to problem number 4065. *The American Mathematical Monthly* 51:169–171, 1944.

[6] P. Hansen, J.-F. Thisse, and R. E. Wendell. Equivalence of solutions to network location problems. *Math. Op. Res.* 11(4):672–678, 1986.

[7] P. Hansen and J.-F. Thisse. Outcomes of voting and planning: Condorcet, Weber and Rawls locations. *Journal of Public Economics* 16(1):1–15, 1981.

[8] D. Kress and E. Pesch. Sequential competitive location on networks. *Europ. J. Op. Res.* 217(3):483–499, 2012.

[9] W.-Y. Lin, Y.-W. Wu, H.-L. Wang and K.-M. Chao. Forming plurality at minimum cost. In *Proc. 9th Int. Workshop Alg. Comput. (WALCOM)*, LNCS 8973, pages 77–88, 2015.

[10] R.D. McKelvey and R.E .Wendell. Voting equilibria in multidimensional choice spaces. *Math. Op. Res.* 1(2):144–158, 1976.

[11] J. W. Tukey. Mathematics and the picturing of data. In *Proc. Int. Cong. Mathematicians*, volume 2, pages 523–531, 1975.

[12] R.E. Wendell and R.D. McKelvey. New perspectives in competitive location theory. *Europ. J. Op. Res.* 6(2):174–182, 1981.

[13] Y.-W. Wu, W.-Y. Lin, H.-L. Wang and K.-M. Chao. Computing plurality points and Condorcet points in Euclidean space. In *Proc. 24th Int. Symp. Alg. Comput. (ISAAC)*, LNCS 8283, pages 688–698, 2013.

On the space of Minkowski summands of a convex polytope

Ioannis Z. Emiris * Anna Karasoulou * Eleni Tzanaki† Zafeirakis Zafeirakopoulos‡

Abstract

We present an algorithm for computing all Minkowski Decompositions (MinkDecomp) of a given convex, integral d-dimensional polytope, using the cone of combinatorially equivalent polytopes. An implementation is given in SAGE.

1 Introduction

Let $A \in \mathbb{Z}^{m\times d}$ be a matrix whose row vectors $a_i \in \mathbb{Z}^d$ positively span $\mathbb{R}^d$. For $b \in \mathbb{R}^m$ the set

$$P_b = \{x \in \mathbb{R}^d \ : \ Ax \leq b\}$$

is a polytope. The set of all *non-empty polytopes* P_b arising this way can be parameterized by their right-hand side vectors b. Let us denote the set of such right hand side vectors b by

$$U(A) = \{b \in \mathbb{R}^m \ : \ P_b \neq \emptyset\}. \qquad (1)$$

Problem 1 ***Minkowski Summands.*** *Given $A \in \mathbb{Z}^{m\times d}$ and $b \in \mathbb{R}^m$, such that $Ax \leq b$ is the H-representation of a convex integral polytope P_b, compute all integral MinkDecomp of P_b.*

In the classical problem of MinkDecomp, which is NP-complete, we are seeking a pair of polytopes whose Minkowski sum equals the input polytope. In this work, we compute instead all possible Minkowski summands. In the first step, we compute the cone of combinatorially equivalent polytopes $U(A)_b$, a subcone of $U(A)$ whose rays and lines generate all the Minkowski summands of P_b. Then, we appropriately shift these rays so that they correspond to integer Minkowski summands. We give an algorithm and its implementation in SAGE [9] performing the computation of all Minkowski summands in any dimension d, extending ideas from [5].

*Department of Informatics & Telecommunications, University of Athens, Athens, Greece. First two authors are partially supported from project "CGL", which acknowledges the financial support of the Future and Emerging Technologies program within the 7th Framework Program for research of the European Commission, under FET-Open grant number: 255827. The authors would like to thank the anonymous reviewers for valuable comments. emiris,akarasou@di.uoa.gr

†Department of Applied Mathematics University of Crete, Crete, Greece, etzanaki@tem.uoc.gr

‡Institute of Information Technologies, Gebze Technical University, Kocaeli, Turkey, zafeirakopoulos@gtu.edu.tr

We focus on the integral decomposition of polytopes. The integral decomposition of polytopes has applications in various areas of mathematics such as integer and mixed integer programming [5], polynomial factorization [4] or implicitization [3]. Since it may happen that an integral polytope has a rational but not an integral decomposition, such a distinction does make sense. Although, qualitatively, a dilation resolves this problem, in many applications, e.g., factorization of polynomials, such a step is not allowed.

Previous work on MinkDecomp algorithms mainly focuses in low dimension [2, 3, 4]. The problem of computing a Minkowski summand in general dimension is reduced to the feasibility of a linear program [6], thus deciding if a polytope is decomposable in order to test polynomial irreducibility. In [1, 5] is explored the cone of combinatorially equivalent polytopes and its computational aspects. Some classical work on polytope decomposition is presented in [7].

2 Computing the Space of Minkowski Summands

A system of inequalities $Ax \leq b$ is *feasible* if it has a solution. Feasibility is characterized by Farkas' lemma.

Lemma 1 (Farkas 1894) *The system of inequalities $Ax \leq b$ is feasible if and only if $y^\top b \geq 0$ for each $y \geq 0$ with $A^\top y = 0$.*

The dual, $U^*(A) = \{y \in \mathbb{R}^m : y^\top b \geq 0 \ \forall b \in U(A)\}$, in view of Lemma 1 becomes

$$U^*(A) = \{y \in \mathbb{R}^m : A^\top y = 0 \text{ and } y \geq 0\}. \qquad (2)$$

It is immediate from Equation (2) that $U^*(A)$ is the intersection of $\ker(A^\top)$ with the positive orthant $\mathbb{R}^m_+$ of $\mathbb{R}^m$. Therefore, $U^*(A)$ is a cone and its primal set $U(A)$ is a cone as well and both contain the origin.

Throughout we will use the following example.

Example Consider the matrix $A \in \mathbb{Z}^{10\times 3}$ and the vector $b \in \mathbb{Z}^{10}$

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \leq \begin{bmatrix} 4 \\ 4 \\ 3 \\ 3 \\ 0 \\ 2 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad (3)$$

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

defining the polytope in Figure 1.

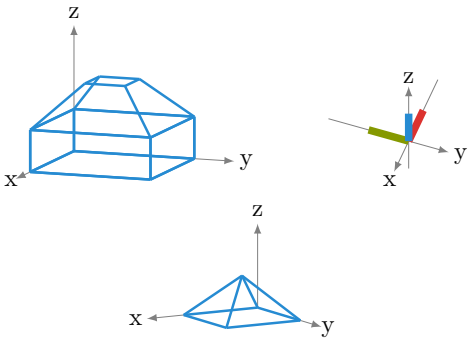


Figure 1: The polytope defined by System (3) and its 2 Minkowski summands.

The inequalities defining the cone $U(A)$ are:

$b_5 + b_6 \geq 0$	$b_4 + b_5 + b_8 \geq 0$	$b_2 + 2b_5 + b_{10} \geq 0$
$b_4 + b_7 \geq 0$	$b_4 + b_5 + b_{10} \geq 0$	$b_2 + b_5 + b_9 \geq 0$
$b_4 + b_5 + b_8 \geq 0$	$b_1 + b_5 + b_7 \geq 0$	$b_1 + 2b_5 + b_8 \geq 0$

Switching from the H-representation to its V-representation, the cone $U(A)$ is generated by 9 rays and 3 lines in $\mathbb{Z}^{10}$.

The *normal cone* of a face F of a polytope P in $\mathbb{R}^d$ is the set

$$\mathcal{N}(F;P) = \{v \in \mathbb{R}^d \ : \ v^\top x = h(P,v) \text{ for all } x \in F\}.$$

The dimension of the normal cone of a k-dimensional face is $(d-k)$. The *normal fan* $\mathcal{N}(P)$ *of* P, which is the collection of the normal cones of all faces of P, is a complete fan in $\mathbb{R}^d$.

The *support function* of a polytope P in $\mathbb{R}^d$, $h(P,;)$, is defined over all $u \in \mathbb{R}^d$ as $h(P,u) = \max\{u^\top x : x \in P\}$. In geometric terms, the evaluation of the support function at $u \in \mathbb{R}^d$ implies that the hyperplane $H_u : x^\top u = h(P,u)$ contains P in one of its closed halfspaces and $H_u \cap P \neq \emptyset$. We call every such H_u an *active* or *supporting hyperplane* of P.

Definition 1 *Two polytopes P, Q in $\mathbb{R}^d$ are* strongly combinatorially equivalent *if, for all $v \in \mathbb{R}^d$*

$$\dim\{y \in P \ : \ v^\top y = h(P,v)\} = \\ = \dim\{y \in Q \ : \ v^\top y = h(Q,v)\}.$$

If polytopes P, Q have the same defining hyperplanes, as in our setup, their normal fans are related by inclusion, i.e., one fan is a subfan of the other. If, in addition, P, Q are strongly combinatorially equivalent, Definition 1 implies $\mathcal{N}(P) = \mathcal{N}(Q)$. We can therefore say that two polytopes are strongly combinatorially equivalent if and only if they have the same normal fan.

Let us give some definitions related to MinkDecomp. Polytopes P_1, P_2 in $\mathbb{R}^d$ are *homothetic* if $P_1 = \rho P_2 + v$ for some $v \in \mathbb{R}^d$ and $\rho > 0$.

Definition 2 *A polytope P in $\mathbb{R}^d$ is called* (homothetically) decomposable *if two polytopes P_1 and P_2 exist with $P = P_1 + P_2$, where P_i in not homothetic to P for $i \in \{1,2\}$. Otherwise P is* (homothetically) indecomposable.

A polytope P_1 is a summand *of a polytope P (denoted as $P_1 \prec P$) if there exists a scalar $\rho > 0$ and a polytope P_2 such that $P = \rho P_1 + P_2$.*

In view of the definition above, trivial polytopes, i.e., points, are indecomposable.

For $b \in U(A)$, we define the *support vector* η_b of the polytope P_b as

$$\eta_b = (h(P_b, a_1), h(P_b, a_2), \ldots, h(P_b, a_d)) .$$

We note that $\eta_b \in \mathbb{Z}^m$ is the componentwise-least right hand side for which $P_b = P_{\eta_b}$. Let us now define the set

$$U(A)_b := \{\eta_v \ : \ v \in U(A) \text{ such that } P_v \prec P_b\}. \quad (4)$$

In [5, 7, 8], the authors show that $U(A)_b$ is a rational polyhedral subcone of $U(A)$ whose structure and extreme rays convey important information on decomposability.

Theorem 2 [7],[8] The set $U(A)_b := \{\eta_v : v \in U(A) \text{ such that } P_v \prec P_b\}$ is a rational polyhedral subcone of $U(A)$ whose extreme rays correspond to indecomposable polytopes and its interior consists of all b' for which $P_{b'}$ is strongly combinatorially equivalent to P_b.

Since $U(A)_b$ is a subcone of the homogeneous (i.e., defined by linear halfspaces) cone $U(A)$, we wish to express $U(A)_b$ as a set of linear inequalities of type $a^\top v \geq 0$ where $a, v \in \mathbb{R}^m$. These inequalities should be imposed from the feasibility of $Ax \leq v$ but, more importantly, they should incorporate the fact that strong combinatorial equivalence is preserved over all faces as well.

Since each face F of P_b can be viewed as a polytope, we can express it as a set $\{x \in \mathbb{R}^d : A_F\, x \leq b_F\}$ where $A_F \in \mathbb{Z}^{\lambda \times d}$, $b_F \in \mathbb{Z}^\lambda$ and $\lambda \in \mathbb{N}$. In this context, we can define $U(A_F)$ and find its subcone $U(A)_{b_F}$ containing all those $y \in \mathbb{Z}^\lambda$ for which the polytope $\{x \in \mathbb{R}^d : A_F\, x \leq y\}$ is combinatorially equivalent to F. However, without reference to the original polytope P_b, the computation of $U(A_F)_{b_F}$ does not keep track of the restrictions imposed on the elements of $U(A)_b$. This indicates that F should be expressed using equalities and inequalities from the original system $Ax \leq b$.

Example (Cont'd) We will apply the procedure described above on a face of our example. Let us pick the facet F defined by $[1,0,1]^\top [x,y,z] = b_1$. Then the system $A_F x \leq b$ for the facet F is

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 1 \\ -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \leq \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ -b_1 \end{bmatrix} \tag{5}$$

For the polyhedron defined by System (5), we obtain the following H-representation of $U(A_F)$:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tilde{b} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

and by mapping the $\tilde{b}_i$'s back to the corresponding b_i's of the input system (note that $\tilde{b} = (b_0, \ldots, b_9, -b_1)$) we obtain the following two constraints: $b_2 + b_8 \geq 0$ and $b_1 + b_4 + b_8 \geq 0$.

The idea in Algorithm 1 is to repeat the above procedure for every face of the input polytope so that none of them "loses support". Note that visiting each face of P_b is essential. If, for example, in the polytope of Figure 1 the algorithm does not visit the top facet, then some b' in the interior of $U(A)_b$ corresponds to the square pyramid. This happens because no restriction prevents the four top vertices to behave as one. This, however, is not acceptable since the square pyramid is not strongly combinatorially equivalent to P_b. Also, starting with $U(A)$ is necessary, since it determines the orientation of the outer normals of P_b. If in our example we started the algorithm with $U(A) = \emptyset$, then we would get the reverse square pyramid as a summand of the polytope, which is not true.

Using the knowledge of the structure of the cone of combinatorially equivalent polytopes, we can compute all indecomposable Minkowski summands of a given polytope. It is however essential, once we have computed the rays of $U(A)_b$, to read out those which produce non-trivial indecomposable polytopes. This is the content of Proposition 3.

We say that $Ax \leq b$, $A \in \mathbb{Z}^{m\times d}$ is an *irredundant description* of $P_b = \{x : Ax \leq b\}$, if the removal of any of the inequalities of the linear system, results in a different polytope (or polyhedron). Notice that this is stronger than requiring b to be the support vector η_b of P_b. The irredundant description of a full dimensional polytope P_b is unique and each of its inequalities supports P_b along a facet. Thus, if $Ax \leq b$ is an irredundant description of a d-polytope with m facets then $A \in \mathbb{Z}^{m\times d}$.

Below we show that, if the input is an irredundant description of P_b, then it is only the rays of $U(A)_b$ that account for the (in)decomposability of P_b.

Proposition 3 *Assume $P_b = \{x : Ax \leq b\}$, $A \in \mathbb{R}^{m\times d}$ is a d-polytope with m facets. Then, the generating rays $b_1, \ldots, b_k$ of $U(A)_b$ correspond to nontrivial indecomposable polytopes, while the generating lines $\pm c_1, \ldots, \pm c_d$ of $U(A)_b$ correspond to points.*

Combining Proposition 3 and Theorem 2, we deduce that each MinkDecomp of P_b into non-trivial indecomposable polytopes is a sum:

$$P_b = \lambda_1 P_{b_1} + \cdots + \lambda_k P_{b_k} + T \tag{6}$$

where $\lambda_1, \ldots, \lambda_k \geq 0$ and $T = \mu_1 P_{c_1} + \cdots + \mu_d P_{c_d}$, $\mu_1, \ldots, \mu_d \in \mathbb{R}$, is a translation.

Lemma 4 *For each polytope $P_c = \{x \in \mathbb{R}^d : Ax \leq c\}$, $A \in \mathbb{R}^{m\times d}$, $0 \neq c \in \mathbb{R}^d$, such that $Ax \leq c$ is feasible,*

1. *if $Ax \leq -c$ is feasible then P_c is a point*
2. *if $Ax \leq -c$ is not feasible then P_c is a non-trivial polytope or P_c is a point whose description $Ax \leq c$ contains a non-active inequality ($c \neq \eta_c$).*

Proof. Since $Ax \leq c$ is a polytope, feasibility of $Ax \leq -c$ implies the existence of a point beyond all faces of P_c. This cannot happen unless P_c is a point. Arguing as above, we see that point 2 is true when P_c is nontrivial. If, however, P_c is a point, the feasibility of both $Ax \leq \pm c$ fails only if the description $Ax \leq c$ contains a hyperplane that does not support P_c. □

Proof. [Proof of Proposition 3] If a polytope P_{b_i} corresponds to an extreme ray of $U(A)_b$, then $Ax \leq b_i$ is feasible whereas $Ax \leq -b_i$ is not. Since, by definition, the cone $U(A)_b$ contains polytopes all whose inequalities are active, Lemma 4.2 rules out the case where $\dim(P_{b_i}) = 0$. Thus, P_{b_i} is a non-trivial indecomposable summand of P_b. If, on the other hand, a polytope P_{c_i} corresponds to an extreme line of $U(A)_b$, then both $Ax \leq c_i$ and $Ax \leq -c_i$ are feasible. In this case, Lemma 4.1 implies that P_{c_i} is a point. □

If we only want to decide whether P_b is indecomposable, Proposition 3 is simplified as follows.

Corollary 5 *Let $P_b = \{x : Ax \leq b\}$, $A \in \mathbb{Z}^{m\times d}$ be a d-polytope with m facets. Then, P_b is indecomposable if and only if cone $U(A)_b$ has a single generating ray.*

Example (Cont'd) We consider the intersection $I = U(A) \cap_i F_i$ of all cones corresponging to faces F_i of the polytope. We compute the V-representation of $U(A)_b$ and get its rays; I is a 7-dimensional cone, with rays:

b_i	$Ax \leq b_i$	vertex set:
$\pm(1,1,0,0,-1,1,0,1,0,1)$	0-dim	$\{(0,0,\pm 1)\}$
$\pm(1,1,0,0,-1,1,0,1,0,1)$	0-dim	$\{(\pm 1,0,0)\}$
$\pm(1,0,0,1,0,0,-1,-1,0,0)$	0-dim	$\{(0,\pm 1,0)\}$
$(0,0,0,0,0,0,0,1,1,0,0)$	1-dim	$\{(0,0,0),(0,-1,0)\}$
$(0,0,0,0,0,0,0,0,0,1,1)$	1-dim	$\{(0,0,0),(-1,0,0)\}$
$(1,1,0,0,0,0,1,0,1,0,1)$	1-dim	$\{(0,0,0),(0,0,1)\}$
$(0,0,0,0,0,0,1,2,2,2,2)$	2-dim	$\{(0,0,0),(-2,0,0),$ $(0,-2,0),(-2,-2,0),\}$ $(-1,-1,1)\}$

The rays $\pm b_1, \pm b_2, \pm b_3$ correspond to points. The next three rays correspond to line segments and the last ray corresponds to a square pyramid, which are exactly the Minkowski summands of the polytope defined by System (3).

In order to find integer indecomposable summands, the rays of $U(A)_b$ may not suffice since they only convey information about the combinatorial type of a polytope.

To resolve this issue, we find an appropriate integer polytope corresponding to each P_{b_i} in Equation (6). More precisely, we find an integer polytope $P_{b_i'}$, combinatorially equivalent to P_{b_i}, such that for all $0 < \lambda < 1$ and all $v \in \mathbb{R}^d$ the polytope $\lambda P_{b_i'} + v$ is not integer.

The first step is to dilate/shrink P_{b_i} enough, so that we get the "smallest possible" integer polytope corresponding to b_i. This can be achieved in the following way: First ensure that one of the vertices of P_{b_i} is the origin, by translating the polytope if needed. Now consider the vertices $v_j = (\frac{a_{j1}}{b_{j1}}, \ldots, \frac{a_{jd}}{b_{jd}}) \in \mathbb{Q}^d$, $1 \le j \le s$, of P_{b_i}, where each $\frac{a_{jk}}{b_{jk}}$ is in reduced form. Then, define:

$$\gcd(v_1, \ldots, v_s) := \gcd\{a_{jk} : 1 \le j \le s, 1 \le k \le d\},$$
$$\operatorname{lcm}(v_1, \ldots, v_s) := \operatorname{lcm}\{b_{jk} : 1 \le j \le s, 1 \le k \le d\}.$$

It is not hard to see that $P'_{b_i} := \{x : Ax \le \lambda' b_i\}$ where $\lambda' = \lambda'(P_{b_i}) := \frac{\operatorname{lcm}(v_1,\ldots,v_s)}{\gcd(v_1,\ldots,v_s)}$ is an integer polytope with the additional property that for any $0 < \lambda < 1$ the polytope $\lambda P'_{b_i}$ is not.

The second and final step is to find a generating set of integer translations. Rather than repeating the above procedure for the trivial polytopes P_{c_i} in Equation (6), we show that the columns $\tilde{c}_1, \ldots, \tilde{c}_d$ of A form a set of integer translation generators in $U(A)_b$.

Lemma 6 *Let $P_b = \{x : Ax \le b\}$, $A \in \mathbb{Z}^{m\times d}$ be a d-polytope with m facets. For each $1 \le i \le d$ set $\tilde{c}_i := Ae_i$ where $e_1, \ldots, e_d$ is the standard basis of $\mathbb{R}^d$. The polytope $\{x : Ax \le \tilde{c}_i\}$ is the unique point e_i.*

Proof. Since the rows of A positively span $\mathbb{R}^d$, the system $Ax \le 0$ has a unique solution. Thus, the same holds for $Ax \le Ae_i$, with unique solution e_i. □

We therefore use the vectors $\tilde{c}_1, \ldots, \tilde{c}_d \in U(A)_b$ as generators of the integer translations in $\mathbb{R}^d$.

Summarizing, we have the following algorithm:

The above algorithm returns a finite set $b_1, \ldots, b_k \in \mathbb{R}^m$ which, together with $\tilde{c}_1, \ldots, \tilde{c}_d$, produces all MinkDecomp of the input polytope P_b. Thus, each way to write $b = \sum_i \lambda_i b_i + \sum_j \mu_j \tilde{c}_j$ yields a decomposition of P_b as in Equation (6). If we want to find integral decompositions of P_b, then the choices for the above λ_i, μ_j should be integers. This allows only a finite number of decompositions.

Algorithm 1 MinkowskiSummands(A, b)

$H_i^- \leftarrow \{x \in \mathbb{R} \ : \ a_i x \le b_i\}$
$H_i \leftarrow \{x \in \mathbb{R} \ : \ a_i x = b_i\}$
$R \leftarrow$ rays of $\ker\left(A^\top\right) \cap \mathbb{R}^m_+$
$U(A) \leftarrow \left\{x \in \mathbb{R}^m \ : \ r^\top x \ge 0 \text{ for } r \in R\right\}$
$U(A)_b \leftarrow U(A)$
for $k \leftarrow 0 \ldots \dim(P) - 1$ **do**
  for F face with $\dim(F) = k$ **do**
    $I \leftarrow \{i_1, \ldots, i_\ell\} \subseteq [m]$ such that $F \subseteq H_{i_s}$
    $A_F \leftarrow \begin{bmatrix} a_i \\ -a_i \\ a_j \end{bmatrix}$ for $i \in I$ and $j \in [m] \setminus I$
    $R \leftarrow$ rays of $\ker\left(A_F^\top\right) \cap \mathbb{R}^{m+\ell}_+$
    $U(A_F) \leftarrow \left\{\tilde{b} \in \mathbb{R}^{m+\ell} \ : \ r^\top \tilde{b} \ge 0 \text{ for } r \in R\right\}$
    Substitute using $\{\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_{d+\ell}\} = \{b_{i_1}, -b_{i_1}, \ldots, b_{i_\ell}, -b_{i_\ell}, b_{i_{\ell+1}}, \ldots, b_{i_d}\}$
    Compute H-rep of $U(A_F)$ wrt $(b_1, \ldots, b_m)$
    $U(A)_b \longleftarrow U(A)_b \cap U(A_F)$
$R \leftarrow$ rays of $U(A)_b$
Summands=$\emptyset$
for r_i in R **do**
  Ensure the origin is a vertex of P_{r_i}
  Compute the vertices $(\frac{a_{j1}}{b_{j1}}, \ldots, \frac{a_{jd}}{b_{jd}})$ of P_{r_i}
  $\lambda' \leftarrow \frac{\operatorname{lcm}(v_1,\ldots,v_s)}{\gcd(v_1,\ldots,v_s)}$
  Summands $\leftarrow \lambda' r_i$
return Summands

References

[1] E. Eisenschmidt, R. Hemmecke, and M. Köppe. Computation of atomic fibers of z-linear maps. *Contributions to Discrete Mathematics*, 6(2), 2011.

[2] I.Z. Emiris and E. Tsigaridas. Minkowski decomposition of convex lattice polygons. In M. Elkadi, B. Mourrain, and R. Piene, eds, *Algebraic Geometry and Geometric Modeling*, pages 217–236. Springer, 2006.

[3] I.Z. Emiris, C. Konaxis, and Z. Zafeirakopoulos. Minkowski decomposition and geometric predicates in sparse implicitization. In *Proc. ACM Intern. Symp. Symb. Alg. Comp.*, pages 157–164, 2015.

[4] S. Gao and A. Lauder. Decomposition of polytopes and polynomials. *Discr. & Comp. Geometry*, 26:89–104, 2001.

[5] M. Henk, M. Köppe, and R. Weismantel. Integral decompositions of polyhedra and some applications in mixed integer programming. *Math. Program., Ser. B*, 94:93–206, 2003.

[6] D. Kesh and S. Mehta. Polynomial irreducibility testing through Minkowski summand computation. In *Proc. Canad. Conf. Comp. Geom.*, pages 43–46, 2008.

[7] P. McMullen. Representations of polytopes and polyhedral sets. *Geometriae Dedicata*, 2:83–99, 1973.

[8] W. Meyer. Indecomposable polytopes. *Trans. Amer. Math. Soc.*, 190:77–86, 1974.

[9] W. Stein et al. *Sage Mathematics Software*. The Sage Development Team.

Approximating the Simplicial Depth in High Dimensions

Peyman Afshani* Donald R. Sheehy† Yannik Stein‡

Abstract

Let P be a set of n points in d-dimensions. The simplicial depth $\sigma_P(q)$ of a point q is the number of d-simplices with vertices in P that contain q in their convex hulls. The simplicial depth is a notion of data depth with many applications in robust statistics and computational geometry. Computing the simplicial depth of a point is known to be a challenging problem. The trivial solution requires $O(n^{d+1})$ time whereas it is generally believed that one cannot do better than $O(n^{d-1})$.

We present two approximation algorithms for computing the simplicial depth of a point in high dimensions with different worst-case scenarios. By combining these approaches, we can compute a $(1+\varepsilon)$-approximation of the simplicial depth in time $\widetilde{O}(n^{d/2+1})$ with high probability ignoring polylogarithmic factors. Furthermore, we present a simple strategy to compute the simplicial depth exactly in $O(n^d \log n)$ time, which provides the first improvement over the trivial $O(n^{d+1})$ time algorithm for $d > 4$. Finally, we show that computing the simplicial depth exactly is #P-complete and W[1]-hard if the dimension is part of the input.

1 Introduction

Let $P \subset \mathbb{R}^d$ be a point set and $q \in \mathbb{R}^d$ be a point. The *simplicial depth* [14] $\sigma_P(q)$ of q with respect to P is the number of subsets $P' \subseteq P$, $|P'| = d+1$, that contain q in their convex hull (see also [4] for an alternate definition). This is one of the important definitions of data depth and has generated interest in both robust statistics and computational geometry since its introduction. Designing efficient algorithms to compute (or approximate) the simplicial depth of a point remains an intriguing task in this area.

Computing the simplicial depth of a single point in 2D was considered even before its formal definition [11] almost three decades ago, perhaps because it translates into an "intuitive" problem of counting the number of triangles containing a given point. In fact, at least three independent papers study this problem in 2D and show how to compute the simplicial depth in $O(n \log n)$ time [9, 11, 14]. This running time is optimal [1]. In 2003, Burr et al. [4] presented an alternate definition for the simplicial depth to overcome some unpleasant behaviors that emerge when dealing with degeneracies. Since we will be dealing with approximations, we will assume general position and thus avoid issues with degeneracy. In 3D, the first non-trivial result offered the bound of $O(n^2)$ [9] but it was flawed; fortunately, the running time of $O(n^2)$ could still be obtained with proper modifications [7]. The same authors presented an algorithm with running time of $O(n^4)$ in 4D. For dimensions beyond 4 there seems to be no significant improvements over the trivial $O(n^{d+1})$ brute-force solution. Furthermore, it is natural to conjecture that computing the simplicial depth should require $\Omega(n^{d-1})$ time: given a set P of n points, it is generally conjectured that detecting whether or not $d+1$ points lie on a hyperplane requires $\Omega(n^d)$ time [8] and this conjecture would imply that detecting whether d points of P and a fixed point q lie on a hyperplane should require $\Omega(n^{d-1})$ time. This is one motivation to consider the approximate version of the problem. In fact, Burr et al. [4] have already expressed interest in computing an approximation to the simplicial depth and they propose a potential approach, although without any worst-case analysis [3].

Here, we only consider relative approximation; additive approximation (with additive error of εn^{d+1}) can be obtained using ε-nets and ε-approximations (see [5, 2] for more details).

Another motivation for computing a relative approximation comes from applications in outlier removal. Intuitively, statistical depth measures how deep a point is embedded in the data cloud with outliers corresponding to points with small values of depth. In such applications, if a small relative error of $(1+\varepsilon)$ is tolerable, then faster outlier removal can be possible using approximations.

2 Approximation in High Dimensions

In this section, we present two approximation algorithms for simplicial depth in high dimensions, each with a different worst case scenario. By combining these strategies, we obtain a constant factor approxi-

*MADALGO, Department of Computer Science, Aarhus University, Denmark, peyman@madalgo.au.dk. Supported in part by the Danish National Research Foundation grant DNRF84 through Center for Massive Data Algorithmics (MADALGO).

†University of Connecticut, USA, don.r.sheehy@gmail.com.

‡Institut für Informatik, Freie Universität Berlin, Germany, yannik.stein@fu-berlin.de. Supported by the Deutsche Forschungsgemeinschaft within the research training group "Methods for Discrete Structures" (GRK 1408).

mation algorithm with $\tilde{O}(n^{d/2+1})$ running time.

2.1 Small Simplicial Depth: Enumeration

Let $P \subset \mathbb{R}^d$ be a set and $q \in \mathbb{R}^d$ a query point. We denote with Δ_P the set of all d-simplices with vertices in P. If $\sigma_P(q)$ is small, a simple counting approach that iterates through all simplices $\Delta \in \Delta_P$ leads to an efficient algorithm. The key is to construct a graph that contains exactly one node per simplex $\Delta \in \Delta_P$. Then, counting can be carried out by a breadth-first search and we avoid looking at subsets of P that do not contain q in their convex hull. For this, we use the Gale transform to dualize the problem. We shortly restate important properties of the Gale transform. For more details see [13]. Let in the following $\mathbf{0}$ denote the origin.

Lemma 1 *Let $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ be a point set with $\sigma_P(\mathbf{0}) > 0$. Then, there is a set $\bar{P} = \{\bar{p}_1, \ldots, \bar{p}_n\} \subset \mathbb{R}^{n-d-1}$ such that a $(d+1)$-subset $P' \subseteq P$ contains $\mathbf{0}$ in its convex hull iff $\bar{P} \setminus \{\bar{p}_i \mid p_i \in P'\}$ defines a facet of* $\mathrm{conv}(\bar{P})$.

Consider now the graph $G_P(q) = (V, E)$ with $V = \Delta_P$. Two simplices Δ, Δ' are adjacent iff Δ' can be obtained from Δ by swapping one point in Δ with a different point in P. We call $G_P(q)$ the *simplicial graph* of P with respect to q.

Lemma 2 *Let $P \subset \mathbb{R}^d$ be a set of size n. Then, $G_P(q)$ is $(n-d-1)$-connected and $(n-d-1)$-regular.*

Proof. We assume w.l.o.g. that $q = \mathbf{0}$. Let Δ, Δ' be two adjacent nodes in $G_P(q)$. Furthermore let $\bar{P}$ denote the Gale transform of P. Set $\bar{\Delta} = \{\bar{p} \mid p \in P \setminus \Delta\}$ and $\bar{\Delta}' = \{\bar{p} \mid p \in P \setminus \Delta'\}$. By Lemma 1, the two sets $\bar{\Delta}$ and $\bar{\Delta}'$ define facets of $\mathrm{conv}(\bar{P})$. Since Δ and Δ' are adjacent, we have $|\Delta \cap \Delta'| = d$ and hence $|\bar{\Delta} \cap \bar{\Delta}'| = n - d - 2$. Thus, the facets defined by $\bar{\Delta}$ and $\bar{\Delta}'$ share a ridge. Hence, $G_P(q)$ is isomorph to the 1-skeleton of the polytope dual to $\mathrm{conv}(\bar{P})$. In particular, this implies that $G_P(q)$ is $(n-d-1)$-connected. It remains to show that the graph is $(n-d-1)$-regular. Let $\Delta \in V$ be a node. It is easy to see that each of the $n-d-1$ points in $P \setminus \Delta$ can be swapped in, each time resulting in a distinct simplex. □

Since $G_P(q)$ is connected, we can count the number of vertices using BFS.

Lemma 3 *Let $P \subset \mathbb{R}^d$ be a set of size n and $q \in \mathbb{R}^d$ a query point. Then, $\sigma_P(q)$ can be computed in $O(n\sigma_P(q))$ time.*

2.2 Large Simplicial Depth: Sampling

If the simplicial depth is large, the enumeration approach becomes infeasible. In this case we apply a simple random sampling algorithm.

Lemma 4 *Let $P \subset \mathbb{R}^d$ be a set and $q \in \mathbb{R}^d$ a query point. Furthermore, let $\varepsilon, \delta > 0$ be constants and let $m \in \mathbb{N}$ be a parameter. If $\sigma_P(q) \geq m$, then $\sigma_P(q)$ can be $(1+\varepsilon)$-approximated in $\tilde{O}(n^{d+1}/m)$ time with error probability $O(n^{-\delta})$.*

Proof. Let $\Delta_1, \ldots, \Delta_k$ be k random $(d+1)$-subsets of P for $k = \left\lceil \frac{4\delta n^{d+1} \log n}{\varepsilon^2 m} \right\rceil$. For each random subset Δ_i, let X_i be 1 iff $q \in \mathrm{conv}(\Delta_i)$ and 0 otherwise. We have $\mu = \mathbb{E}[\sum_{i=1}^k X_i] = k\frac{\sigma_P(q)}{n^{d+1}} = \frac{4\delta\sigma_P(q)\log n}{\varepsilon^2 m} \geq \frac{4\delta}{\varepsilon^2}\log n$. Applying the Chernoff bound, we get $\Pr[|\sum_{i=1}^k X_i - \mu| \geq \varepsilon\mu] = O(n^{-\delta})$. Thus, $\frac{n^{d+1}}{k}X$ is a $(1+\varepsilon)$-approximation of $\sigma_P(q)$ with error probability $O(n^{-\delta})$.

For $d = O(1)$, we can test in $O(1)$ whether a given $(d+1)$-subset of P contains a point in its convex hull. Hence, the running time is dominated by the number of samples. □

2.3 Combining the Strategies

Theorem 5 *Let $P \subset \mathbb{R}^d$ be a set and $q \in \mathbb{R}^d$ a query point. Furthermore, let $\varepsilon > 0$ and $\delta > 0$ be constants. Then, $\sigma_P(q)$ can be $(1+\varepsilon)$-approximated in $\tilde{O}(n^{d/2+1})$ time with error probability $O(n^{-\delta})$.*

Proof. We apply the algorithm from Lemma 3 and stop it once $n^{d/2}$ nodes of $G_P(q)$ are explored. This requires $O(n^{d/2+1})$ time. If the graph is not yet fully explored, we know $\sigma_P(q) \geq n^{d/2}$. We can now apply the algorithm from Lemma 4 and compute a $(1+\varepsilon)$-approximation in $\tilde{O}(n^{d/2+1})$ time with error probability $O(n^{-\delta})$. □

3 An Exact Algorithm in High Dimensions

In this section we describe a simple strategy to compute the simplicial depth exactly in $O(n^d \log n)$ time. While we do not achieve the conjectured lower bound of $\Omega(n^{d-1})$, we cut down roughly a factor n compared to the trivial upper bound of $O(n^{d+1})$. Note that this almost matches the best previous bound of $O(n^4)$ in 4D as well [7].

W.l.o.g, assume q is the origin, $\mathbf{0}$. Our main idea is very simple: consider d points $p_1, \ldots, p_d \in P$. Let $\overrightarrow{r_i}$ be the ray that originates from $\mathbf{0}$ towards $-p_i$. We would like to count how many points $p \in P$ can create a simplex with $p_1, \ldots, p_d$ that contains $\mathbf{0}$. We observe that this is equivalent to counting the number of points of P that lie inside the simplex created by rays

$\overrightarrow{r_1}, \ldots, \overrightarrow{r_d}$. We can count this number in polylogarithmic time if we spend $\tilde{O}(n^d)$ time to build a simplex range counting data structure on P. This would give an algorithm with overall running time of $\tilde{O}(n^d)$. We can cut the log factors down to one by employing a slightly more intelligent approach.

We use the following observation made by Gil et al. [9].

Observation 1 *Let q be a point inside a simplex $a_1 \ldots a_{d+1}$ and let a'_i be a point on the ray $\overrightarrow{qa_i}$. Then, $q \in \text{conv}\{a_1, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_{d+1}\}$.*

Pick two arbitrary parallel hyperplanes h_1 and h_2 such that P lies between them. This can be done easily in $O(n)$ time. Next, using central projection from $\mathbf{0}$, we map the points onto the hyperplanes h_1 and h_2: for every point $p_i \in P$, we create the ray $\overrightarrow{\mathbf{0}p_i}$ and let p'_i be the intersection of the ray with h_1 or h_2. Thus, the point set P can be mapped to two point sets P_1 and P_2 where P_1 lies on h_1 and p_2 lies on P_2 and furthermore, by Observation 1, $\sigma_P(q) = \sigma_{P_1 \cup P_2}(q)$.

Now we use the following result from the simplex range counting literature.

Theorem 6 *[6] Given a set of n points in d-dimensional space, and any constant $\varepsilon > 0$, one can build a data structure of size $O(n^{d+\varepsilon})$ in $O(n^{d+\varepsilon})$ expected preprocessing time, such that given any query simplex Δ, the number of points in Δ can be counted in $O(\log n)$ time.*

We build the above data structure on P_1 and P_2. However, since both of these point sets lie on a $(d-1)$-dimensional flat, the preprocessing time is $O(n^{d-1+\varepsilon}) = O(n^d)$ if we choose $\varepsilon = 1/2$. Next, for any d tuples of points $p_1, \ldots, p_d$, we create the rays $\overrightarrow{r_1}, \ldots, \overrightarrow{r_d}$ and the corresponding simplex Δ. We find the intersection of Δ in $O(1)$ time with hyperplanes h_1 and h_2 and issue two simplex range counting queries, one in each hyperplane. Thus, in $O(\log n)$ time, we can count how many simplices contain $\mathbf{0}$ that are made by points $p_1, \ldots, p_d$. We add all these numbers over all d tuples, which counts each simplex containing $\mathbf{0}$ exactly $(d+1)$ times. The number of d-tuples is $O(n^d)$ and for each we spend $O(\log n)$ time querying the data structures. Thus, we obtain the following theorem.

Theorem 7 *Given a set P of n points in $\mathbb{R}^d$, the simplicial depth of a point p can be computed in $O(n^d \log n)$ expected time.*

4 Complexity

Let $P \subset \mathbb{R}^d$ be a set of n points and $q \in \mathbb{R}^d$ a query point. If the dimension is constant, then clearly computing $\sigma_P(q)$ can be carried out in polynomial time. We now consider the case that d is part of the input. We show that in this case computing the simplicial depth is #P-complete by a reduction from counting the number of perfect matchings in bipartite graphs.

Theorem 8 *Let $P \subset \mathbb{R}^d$ be a set and $q \in \mathbb{R}^d$ a query point. Then, computing $\sigma_P(q)$ is #P-complete if the dimension is part of the input.*

Proof. Let $G = (V, E)$ be a bipartite graph with $|V| = n$ and $|E| = m$. It is well known that computing the number of perfect matchings in G is #P-complete [15]. Let $\mathcal{P}_H \subset \mathbb{R}^m$ be the perfect matching polytope for G [10, Chapter 30]. It is defined by $m + 2n$ half-spaces. Furthermore, the number of vertices of $\mathcal{P}_H$ equals the number k of perfect matchings in G. Consider now the dual polytope $\mathcal{P}_V \subset \mathbb{R}^m$. It is the convex hull of $m + 2n$ points $P \subset \mathbb{R}^m$ and the number of facets equals k. Let $\bar{P} \subset \mathbb{R}^{2n-1}$ be the Gale transform of P. By Lemma 1, there is a bijection between the facets of $\mathcal{P}_V$ and the $(2n-1)$-simplices with vertices in $\bar{P}$ that contain $\mathbf{0}$ in their convex hull. Hence, $\sigma_{\bar{P}}(\mathbf{0}) = k$. □

Next, we show that computing the simplicial depth is W[1]-hard with respect to the parameter d by a reduction to *d-Carathéodory*. In d-Carathéodory, we are given a set $P \subset \mathbb{R}^d$ and have to decide whether there is a $(d-1)$-simplex with vertices in P that contains $\mathbf{0}$ in its convex hull. Knauer et al. [12] proved that this problem is W[1]-hard with respect to the parameter d.

Theorem 9 *Let $P \subset \mathbb{R}^d$ be a set and $q \in \mathbb{R}^d$ a query point. Then, computing $\sigma_P(q)$ is W[1]-hard with respect to the parameter d.*

Proof. Assume we have access to an oracle that, given a query point q and a set $Q \subset \mathbb{R}^d$, returns $\sigma_Q(q)$. We show that #d-Carathéodory can be decided with two oracle queries.

Let k_d denote the number of $(d-1)$-simplices with vertices in P that contain $\mathbf{0}$ in their convex hulls and let k_{d+1} denote the number of d-simplices with vertices in P that contain $\mathbf{0}$ in their interior. Then $\sigma_P(\mathbf{0})$ can be written as $(|P| - d)k_d + k_{d+1}$. We want to decide whether $k_d > 0$. For each point $p \in P$ let $\tilde{p} \in \mathbb{R}^{d+1}$ denote the $(d+1)$-dimensional point that is obtained by appending a 1-coordinate and similarly, for each subset $P' \subset P$ let $\tilde{P'}$ denote the set $\{\tilde{p} \mid p \in P'\} \subset \mathbb{R}^{d+1}$. We denote with S the set $\{(0, \ldots, 0, -1)^T, (0, \ldots, 0, -2)^T\} \subset \mathbb{R}^{d+1}$ and set $Q = \tilde{P} \cup S$. Again, we want to express $\sigma_Q(\mathbf{0})$ as a function of k_d and k_{d+1}. Let $Q' \subset Q$, $|Q'| = d+2$, be a subset that contains $\mathbf{0}$ in its convex hull. Clearly, Q' has to contain a point from S. Let $\tilde{P'} = Q' \cap \tilde{P}$ denote the part from $\tilde{P}$ and let $S' = Q' \cap S$ denote the part from S. By construction of S, we have

$(0,\ldots,0,1)^T \in \text{conv}(\tilde{P}')$ and hence $\mathbf{0} \in \text{conv}(P')$. That is, each $(d+2)$-simplex with vertices in Q that contains $\mathbf{0}$ in its convex hull corresponds to either a d-simplex or a $(d-1)$-simplex with vertices in P that contains $\mathbf{0}$ in its convex hull. Consider now a set $P' \subset P$ with $|P'| = d+1$ and $\mathbf{0} \in \text{conv}(P)$. Then, the corresponding set $\tilde{P}'$ can be extended in two ways to a subset $Q' \subset Q$, $|Q'| = d+2$, with $\mathbf{0} \in \text{conv}(Q')$ by taking either point in S. On the other hand, if $P' \subset P$ is a subset of size d with $\mathbf{0} \in \text{conv}(P')$, then we can extend $\tilde{P}'$ to a set $Q' \subset Q$, $|Q'| = d+2$, with $\mathbf{0} \in \text{conv}(Q')$ by either taking both points in S or by taking one arbitrary point in $\tilde{P} \setminus \tilde{P}'$ and either point in S. Hence, we have $\sigma_Q(\mathbf{0}) = 2k_{d+1} + k_{d-1} + 2(|P|-d)k_{d-1}$. Since $k_d = \sigma_Q(\mathbf{0}) - 2\sigma_P(\mathbf{0})$, we can decide whether $k_d > 0$ with two oracle queries. □

The following theorem is now immediate.

Theorem 10 *Let $P \subset \mathbb{R}^d$ be a set of d-dimensional points and $q \in \mathbb{R}^d$ a query point. Then, computing $\sigma_P(q)$ is #P-complete and W[1]-hard with respect to the parameter d.*

We conclude the section with a constructive result: although computing the simplicial depth is #P-complete, it is possible to determine the parity in polynomial-time.

Theorem 11 *Let $P \subset \mathbb{R}^d$ be a set of points and $q \in \mathbb{R}^d$ a query point. If $n-d-1$ is odd or $\binom{n}{d}$ is even, then $\sigma_P(q)$ is even. Otherwise, $\sigma_P(q)$ is odd.*

Proof. We assume w.l.o.g. that q is the origin. Since the simplicial graph $G_P(\mathbf{0})$ is $(n-d-1)$-regular, the product $(n-d-1)|V| = (n-d-1)\sigma_P(\mathbf{0})$ is even. If $(n-d-1)$ is odd, $\sigma_P(q)$ has to be even. Assume now $(n-d-1)$ is even. We construct a new point set Q in $\mathbb{R}^{d+1}$ similar as in the proof of Theorem 9. Let R denote the set $\{(0,\ldots,0,-1)^T, (0,\ldots,0,2)^T\} \subset \mathbb{R}^{d+1}$ and set $Q = \tilde{P} \cup R \subset \mathbb{R}^{d+1}$, where $\tilde{P}$ is defined as in the proof of Theorem 9. Let us now consider the graph $G_Q(\mathbf{0})$. Since $n-d-1$ is even, $(|Q| - (d+1) - 1) = n-d$ is odd. Now, $G_Q(\mathbf{0})$ is $(n-d)$-regular and thus $\sigma_Q(\mathbf{0})$ is even. Let $Q' \subset Q$, $|Q| = d+2$, be a subset that contains the origin in its convex hull. Then either (i) $R \subset Q'$ or (ii) Q' contains the point $r = (0,\ldots,0,-1)^T \in R$ and $d+1$ points $\tilde{P}' \subseteq \tilde{P}$ with $(0,\ldots,0,1)^T \in \text{conv}(\tilde{P}')$. There are $\binom{n}{d}$ sets Q' with Property (i) and $\sigma_P(\mathbf{0})$ sets Q' with Property (ii). Hence, we have $\sigma_Q(\mathbf{0}) = \sigma_P(\mathbf{0}) + \binom{n}{d}$ is even and thus $\sigma_P(\mathbf{0})$ is odd iff $\binom{n}{d}$ is odd. □

Acknowledgements. This work was initiated while YS was visiting MADALGO in Aarhus. I would like to thank the working group for their hospitality and for a constructive atmosphere.

References

[1] G. Aloupis, C. Cortés, F. Gómez, M. Soss, and G. Toussaint. Lower bounds for computing statistical depth. *Comput. Stat. & Data Anal.*, 40(2):223–229, 2002.

[2] Amitabha Bagchi, Amitabh Chaudhary, David Eppstein, and Michael T Goodrich. Deterministic sampling and range counting in geometric data streams. *TALG 2007*, 3(2):16.

[3] Michael Burr, Eynat Rafalin, and Diane L. Souvaine. Simplicial depth: An improved definition, analysis, and efficiency for the finite sample case.

[4] Michael Burr, Eynat Rafalin, and Diane L. Souvaine. Simplicial depth: An improved definition, analysis, and efficiency for the finite sample case. In *CCCG*, pages 136–139, 2004.

[5] B. Chazelle. *The discrepancy method: randomness and complexity*. Cambridge University Press, 2000.

[6] Bernard Chazelle, Micha Sharir, and Emo Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992.

[7] A. Y. Cheng and M. Ouyang. On algorithms for simplicial depth. In *CCCG*, pages 53–56, 2001.

[8] Jeff Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal on Computing*, 28(4):1198–1214, 1999.

[9] J. Gil, W. Steiger, and A. Wigderson. Geometric medians. *Discrete Math.*, 108(1):37–51, 1992.

[10] R. L. Graham, M. Grötschel, and L. Lovász. *Handbook of combinatorics*, volume 2. Elsevier, 1995.

[11] S. Khuller and J. S. B. Mitchell. On a triangle counting problem. *Inf. Process. Lett.*, 33(6):319–321, 1990.

[12] C. Knauer, H. R. Tiwary, and D. Werner. On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems. In *STACS 2011*, pages 649–660.

[13] R. Thomas R. *Lectures in Geometric Combinatorics*. American Mathematical Society, 2006.

[14] P. J. Rousseeuw and I. Ruts. Bivariate location depth. *J. R. Stat. Soc. Series C (Appl. Stat.)*, 45(4):516–526, 1996.

[15] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189 – 201, 1979.

Bottleneck Distances and Steiner Trees in the Euclidean d-Space

Stephan S. Lorenzen Pawel Winter *

Abstract

Some of the most efficient heuristics for the Euclidean Steiner minimal trees in the d-dimensional space, $d \geq 2$, use Delaunay tessellations and minimum spanning trees to determine small subsets of geometrically close terminals. Their low-cost Steiner trees are determined and concatenated in a greedy fashion to obtain low cost trees spanning all terminals. The weakness of this approach is that obtained solutions are topologically related to minimum spanning trees. To obtain better solutions, bottleneck distances are utilized to determine good subsets of terminals without being constrained by the topologies of minimum spanning trees. Computational experiments show a significant solution quality improvement.

1 Introduction

Given a set of *terminals* $N = \{t_1, t_2, ..., t_n\}$ in the Euclidean d-dimensional space $\mathcal{R}^d$, $d \geq 2$, the *Euclidean Steiner minimal tree* (ESMT) *problem* asks for a shortest connected network $T = (V, E)$, where $N \subseteq V$. The points in $S = V \setminus N$ are called *Steiner points*. The length $|uv|$ of an edge $(u, v) \in E$ is the Euclidean distance between u and v. The length $|T|$ of T is the sum of the lengths of the edges in T. T must be a tree. It is called the *Euclidean Steiner minimal tree* and is denoted by $\mathrm{SMT}(N)$. Many variants with important applications in the design of transportation and communication networks and in the VLSI design have been investigated. While the ESMT problem is one of the oldest optimization problems, it remains an active research area due to its difficulty, many open questions and challenging applications. The reader is referred to [3] for the fascinating history of the ESMT problem.

The ESMT problem is NP-hard [4]. A good exact method for solving problem instances with up to 50.000 terminals in R^2 is available [9]. However, no analytical method can exist for $d \geq 3$ [1]. Furthermore, no numerical approximation seems to be able to solve instances with more than 15-20 terminals [6]. It is therefore essential to develop good quality heuristics for $d \geq 3$. Several heuristics have been proposed in the literature [15, 7, 10]. In particular, the heuristic suggested in [10] builds on a R^2-heuristic [13]. Both use **D**elaunay tessellations and **M**inimum spanning trees and are therefore referred to as DM-heuristics.

$\mathrm{SMT}(N)$ must have $n - 2$ Steiner points, each incident with 3 edges [8]. Terminals must be incident with exactly 1 edge (possible of zero-length). Non-zero-length edges must meet at Steiner points at angles that are at least 120^o. If a pair of Steiner points s_i and s_j is connected by a zero-length edge, then s_i or s_j is connected via a zero-length edge to a terminal and the three non-zero-length edges incident with s_i and s_j must make 120^o with each other. Any geometric network $\mathrm{ST}(N)$ satisfying the above degree and angle conditions is called a *Steiner tree*. The underlying undirected graph (where the coordinates of Steiner points are immaterial) is called a *Steiner topology* of N. If $\mathrm{ST}(N)$ has no zero-length edges, then it is called a *full Steiner tree*. Every Steiner tree $\mathrm{ST}(N)$ can be decomposed into one or more full Steiner subtrees whose degree one points are either terminals or Steiner points overlapping with terminals.

A reasonable approach to find a good suboptimal solution to the ESMT problem is therefore to identify few subsets $N_1, N_2, ..., N_z$, and their low cost Steiner trees $\mathrm{ST}(N_1), \mathrm{ST}(N_2), ..., \mathrm{ST}(N_z)$, such that a union $\mathrm{ST}(N)$ of some of them will be a good approximation of $\mathrm{SMT}(N)$.

The Delaunay tessellation of N in $\mathcal{R}^d$ is denoted by $\mathrm{DT}(N)$ [2]. It is well-known that a minimum spanning tree of N, denoted by $\mathrm{MST}(N)$, is a subgraph of $\mathrm{DT}(N)$. A face σ of $\mathrm{DT}(N)$ is *covered* if the subgraph of $\mathrm{MST}(N)$ induced by the corners of σ is a tree.

Let $N_\sigma \subseteq N$ denote the corners of a face σ of $\mathrm{DT}(N)$. Let $\mathrm{ST}(N_\sigma)$ denote a Steiner tree spanning N_σ. Let F be a forest whose vertices form a superset of N. Suppose that the terminals of N_σ are in different subtrees of F. The *concatenation* $F \oplus \mathrm{ST}(N_\sigma)$ of F with $\mathrm{ST}(N_\sigma)$ is a forest obtained by adding to F all Steiner points and all edges of $\mathrm{ST}(N_\sigma)$.

Let $T = \mathrm{MST}(N)$. The *contraction* $T \ominus N_\sigma$ of T by N_σ is obtained by replacing the vertices in N_σ by a single vertex n_σ. Cycles in $T \ominus N_\sigma$ are destroyed by removing their longest edges.

The DM-heuristic constructs $\mathrm{DT}(N)$ and $\mathrm{MST}(N)$ in the preprocessing phase. For corners N_σ of every covered face σ of $\mathrm{DT}(N)$, a low cost Steiner tree $\mathrm{ST}(N_\sigma)$ is determined [10]. If full, it is stored in a priority queue Q ordered by non-decreasing Steiner

*Dept. of Computer Science, Univ. of Copenhagen, stephan.lorenzen@gmail.com, pawel@di.ku.dk, full version of the paper: http://www.diku.dk/~pawel/bottleneck.pdf

ratios $\rho(\mathrm{ST}(N_\sigma)) = |\mathrm{ST}(N_\sigma)|/|\mathrm{MST}(N_\sigma)|$. Greedy concatenation, starting with the forest F of isolated terminals in N, is then used to form $\mathrm{ST}(N)$.

The weakness of the DM-heuristic is that it relies on covered faces of $\mathrm{DT}(N)$. The Steiner topology of $\mathrm{ST}(N)$ is therefore dictated by the topology of T. This is a good strategy in many cases but there are also cases where this will exclude good solutions. Con-

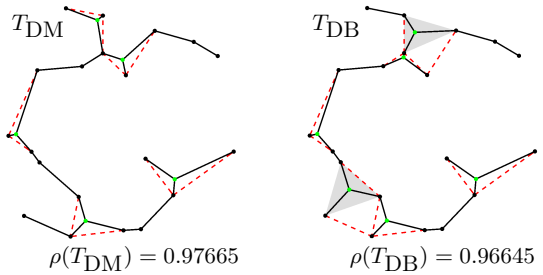


Figure 1: Uncovered faces of $\mathrm{DT}(N)$ can improve solutions. Edges of T not in Steiner trees are red.

sider for example the two Steiner trees in Fig. 1. Only covered faces of $\mathrm{DT}(N)$ are considered in T_{DM}. By considering some uncovered faces (shaded), a better Steiner tree T_{DB} can be obtained.

We wish to detect useful uncovered faces and include them into the greedy concatenation. However, some uncovered faces of $\mathrm{DT}(N)$ can be harmful in the greedy concatenation even though they seem to be useful locally. As illustrated in Fig. 2, use of the uncovered face σ of $\mathrm{DT}(N)$ in $\mathcal{R}^2$ with the set of corners $N_\sigma = \{t_i, t_j, t_k\}$ will lead to a Steiner tree $\mathrm{ST}(N)$ longer than T while the ratio $\rho(\mathrm{ST}(N_\sigma))$ is lowest among all faces of $\mathrm{DT}(N)$.

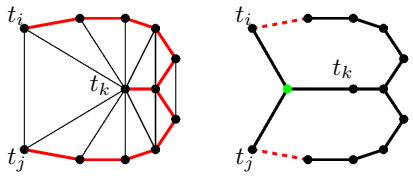


Figure 2: $\rho(\mathrm{ST}(N_\sigma))$ for $N_\sigma = \{t_i, t_j, t_k\}$ is low but the inclusion of $\mathrm{ST}(N_\sigma)$ increases its length beyond $|T|$.

2 DB-Heuristic in $\mathcal{R}^d$

The *bottleneck distance* $|t_i t_j|_T$ between two terminals $t_i, t_j \in N$ is the length of the longest edge on the path from t_i to t_j in $T = \mathrm{MST}(N)$. Note that $|t_i t_j|_T = |t_i t_j|$ if $(t_i, t_j) \in T$.

The *bottleneck minimum spanning tree* $B_T(N_\sigma)$ of a set of points $N_\sigma \subseteq N$ is defined as the minimum spanning tree of the complete graph with N_σ as its vertices and with $|t_i t_j|_T$ being the cost of an edge $(t_i, t_j), t_i, t_j \in N_\sigma$. If N_σ is covered by T, then $|B_T(N_\sigma)| = |\mathrm{MST}(N_\sigma)|$.

Consider a Steiner tree $\mathrm{ST}(N_\sigma)$ spanning $N_\sigma \subseteq N$. Let the *bottleneck Steiner ratio* $\beta_T(\mathrm{ST}(N_\sigma)) = |\mathrm{ST}(N_\sigma)|/|B_T(N_\sigma)|$ If N_σ is covered by T, then $\beta_T(\mathrm{ST}(N_\sigma)) = \rho(\mathrm{ST}(N_\sigma))$.

The DB-heuristic constructs the **D**elaunay tesselation $\mathrm{DT}(N)$ and uses T to determine **B**ottleneck distances. For corners N_σ of each k-face σ of $\mathrm{DT}(N)$, $2 \le k \le d+1$, a low cost Steiner tree $\mathrm{ST}(N_\sigma)$ is determined using a heuristic [10]. Each full $\mathrm{ST}(N_\sigma)$ is stored in a priority queue Q_B ordered by non-decreasing bottleneck Steiner ratios. If σ is a 1-face, then $\mathrm{ST}(N_\sigma)$ is the edge connecting the two corners of σ. Such $\mathrm{ST}(N_\sigma)$ is added to Q_B only if it is an edge in T.

Let F be the forest of isolated terminals from N. A greedy concatenation is then applied repeatedly until F becomes a tree. Let $\mathrm{ST}(N_\sigma)$ be a Steiner tree with *currently* smallest bottleneck Steiner ratio in Q_B. If any pair of terminals in N_σ is connected in F, $\mathrm{ST}(N_\sigma)$ is discarded. Otherwise, $F = F \oplus \mathrm{ST}(N_\sigma)$ and $T = T \ominus N_\sigma$, see Fig. 3. Such contraction of T may reduce bottleneck distances between up to $O(n^2)$ pairs of terminals. Hence, bottleneck Steiner ratios of some Steiner trees still in Q_B need to be updated, preferrably in a lazy fashion, see Section 3.

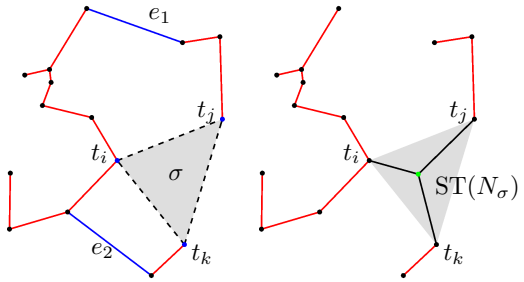


Figure 3: The insertion of $ST(N_\sigma)$, $N_\sigma = \{t_i, t_j, t_k\}$

3 Contractions and Bottleneck Distances

As face-spanning Steiner trees are added to F, their corners are contracted in the current minimum spanning tree T. Contractions will reduce bottleneck distances between some pairs of terminals. As a consequence, bottleneck Steiner ratios of face-spanning Steiner trees still in Q_B will increase. A face-spanning Steiner tree subsequently extracted from Q_B will not necessarily have the smallest bottleneck Steiner ratio unless Q_B has been rearranged or appropriate lazy updating is carried out.

Steiner trees of faces of $\mathrm{DT}(N)$ are extracted from Q_B one by one. A face σ is discarded if some of its

corners are already connected in F. The bottleneck Steiner ratio $\beta_T(\mathrm{ST}(N_\sigma))$ of the extracted Steiner tree $\mathrm{ST}(N_\sigma)$ may have changed since the last time $\mathrm{ST}(N_\sigma)$ was pushed onto Q_B. Hence, $\beta_T(\mathrm{ST}(N_\sigma))$ has to be recomputed. If it increased, $\mathrm{ST}(N_\sigma)$ is pushed back onto Q_B (with the new bottleneck Steiner ratio). If not, $\mathrm{ST}(N_\sigma)$ is used to update F and to contract T.

A modified version of a *dynamic rooted tree* [12] to maintain a changing minimum spanning tree T (caused by contractions) and to answer bottleneck distance querries has been used. When using balanced binary trees to implement dynamic rooted trees, a bottleneck distance query takes $O((\log n)^2)$ amortized time. Since only faces of $\mathrm{DT}(N)$ are considered, a contraction takes $O((d \log n)^2)$ time.

4 Computational results

The DB-heuristic was compared with the DM-heuristic. Both Steiner ratios and CPU times were examined. To get reliable comparisons, they were averaged over several runs. Furthermore, the results in $\mathcal{R}^2$ were compared with the results achieved by the exact GeoSteiner algorithm [9].

The DM- and DB-heuristics were implemented in C++ [1] and run on a Lenovo ThinkPad S540 with a 2 GHz Intel Core i7-4510U processor and 8 GB RAM.

Both heuristics were tested on randomly generated problem instances in $\mathcal{R}^d$, $d = 2, 3, ..., 6$, as well as on library problem instances. Randomly generated instances were points uniformly distributed in $\mathcal{R}^d$-hypercubes.

The library problem instances consisted of the benchmark instances from the 11-th DIMACS Challenge [5]. For comparing the DB-heuristic with the GeoSteiner algorithm, ESTEIN instances in $\mathcal{R}^2$ were used [5].

The new DB-heuristic outperforms the DM-heuristic by $0.2-0.3\%$ for $d = 2$, $0.4-0.5\%$ for $d = 3$, $0.6-0.7\%$ for $d = 4$, $0.7-0.8\%$ for $d = 5$ and $0.8-0.9\%$ for $d = 6$. This is a significant improvement for the ESMT problem as will be seen below, when comparing $\mathcal{R}^2$-results to the optimal solutions obtained by the exact GeoSteiner algorithm [9].

CPU times for the DM- and DB-heuristics for $d = 2, 3, ..., 6$, are shown in Fig. 4. It can be seen that the improved quality comes at a cost for $d \geq 4$. This is due to the fact that the DB-heuristic constructs low cost Steiner trees for all $O(n^{\lceil d/2 \rceil})$ faces of $\mathrm{DT}(N)$ [11] while the DM-heuristic does it for covered faces only.

Fig. 5 shows how DB-, DM-heuristic and GeoSteiner (GS) performed on ESTEIN instances in $\mathcal{R}^2$. Steiner ratios and CPU times averaged over all 15 ESTEIN instances of the given size, except for

[1] The DB-heuristic code and instructions on how to run it can be found at `https://github.com/StephanLorenzen/ESMT-heuristic-using-bottleneck-distances`

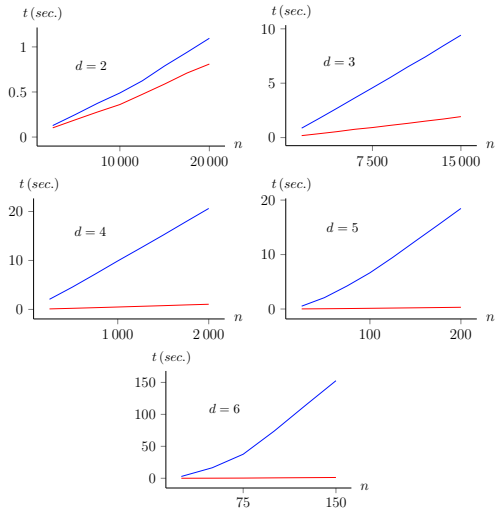


Figure 4: CPU times for DM (red) and DB (blue), $d = 2, 3, ..., 6$.

$n = 10.000$ which has only one instance. It can be seen that the DB-heuristic produces better solutions than the DM-heuristic without any significant increase of the CPU time. It is also worth noticing that the DB-heuristic gets very close to the optimal solutions. This may indicate that the DB-heuristic also produces high quality solutions when $d > 2$, where optimal solutions are only known for instances with at most 20 terminals.

Figure 5: Averaged ratios and CPU times for ESTEIN instances in $\mathcal{R}^2$. DM (red), DB (blue), GeoSteiner (green).

The results for ESTEIN instances in $\mathcal{R}^3$ are presented in Fig. 6. The green plot for $n = 10$ is the average ratio and CPU time of the exact solutions [14]. Once again, the DB-heuristic outperforms the DM-heuristic when comparing the quality of solutions. However, the running times are now up to four times worse.

The DB-heuristic starts to struggle when $d \geq 4$.

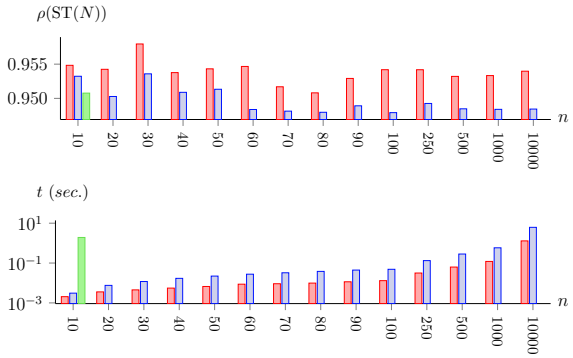
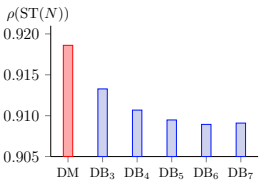


Figure 6: Averaged ratios and CPU times for ESTEIN instances in $\mathcal{R}^3$. DM (red), DB (blue), exact (green).

This is caused by the number of faces of DT(N) for which low cost Steiner trees must be determined. The DB-heuristic was therefore modified to consider only faces with less than k terminals, for $k = 3, 4, ..., d+1$. Fig. 7 shows the performance of this modified DB$_k$-heuristic for $k = 3, 4, ..., 7$, on a set with 100 terminals in $\mathcal{R}^6$. Note that DB$_7$ = DB.



Method	t
BM	0.4714
DB$_3$	0.6000
DB$_4$	6.0525
DB$_5$	26.2374
DB$_6$	51.3653
DB$_7$ = DB	62.8098

Figure 7: DB$_k$ for $k = 3, 4, ..., 7$, $d = 6$ and $n = 100$.

As expected, the DB$_k$-heuristic runs much faster when larger faces of DT(N) are disregarded. Already the DB$_4$-heuristic seems to be a reasonable alternative since solutions obtained by DB$_k$-heuristic, $5 \leq k \leq 7$ are not significantly better.

5 Summary and conclusion

Computational results show a significant improvement in the quality of the Steiner trees produced by the DB-heuristic where the topologies of the solutions are no longer constrained by minimum spanning trees. Its CPU times are comparable to the CPU times of the DM-heuristic in $\mathcal{R}^d$, $d = 2, 3$. It runs slower for $d \geq 4$. However, CPU times can be significantly improved by skipping larger faces of DT(N). This results in only small decrease of quality of solutions obtained.

References

[1] C. Bajaj, The algebraic degree of geometric optimization problems, *Discrete and Computational Geometry* **3** (1988), 177–191.

[2] M. de Berg, O. Cheong, M. van Krevald, and M. Overmars, *Computational Geometry - Algorithms and Applications* (3. ed.), Springer (2008).

[3] M. Brazil, R. Graham, D. Thomas, and M. Zachariasen, On the history of the Euclidean Steiner tree problem, *Archive for History of Exact Sciences* **68** (2014), 327–354.

[4] M. Brazil and M. Zachariasen, *Optimal Interconnection Trees in the Plane*, Springer (2015).

[5] DIMACS and ICERM, *11th DIMACS Implementation Challenge: Steiner Tree Problems*, `http://dimacs11.cs.princeton.edu/` (2014).

[6] M. Fampa, J. Lee, and N. Maculan, An overview of exact algorithms for the Euclidean Steiner tree problem in n-space, *Int. Trans. in OR* (2015).

[7] V. L. do Forte, F. M. T. Montenegro, J. A. de Moura Brito, and N. Maculan, Iterated local search algorithms for the Euclidean Steiner tree problem in n dimensions, *Int. Trans. in OR* (2015).

[8] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland (1992).

[9] D. Juhl, D. M. Warme, P. Winter, and M. Zachariasen, The GeoSteiner software package for computing Steiner trees in the plane: An updated computational study, *Proc. of the 11th DIMACS Implementation Challenge* (2014). `http://dimacs11.cs.princeton.edu/workshop.html`

[10] A. Olsen, S. Lorenzen, R. Fonseca, and P. Winter, Steiner tree heuristics in Euclidean d-space, *Proc. of the 11th DIMACS Implementation Challenge* (2014). `http://dimacs11.cs.princeton.edu/workshop.html`

[11] R. Seidel, The upper bound theorem for polytopes: an easy proof of its asymptotic version, *Comp. Geom.-Theor. Appl.* **5** (1995), 115–116.

[12] D. D. Sleator and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. and Syst. Sci.* **26**, 3 (1983), 362–391.

[13] J. M. Smith, An O($n \log n$) heuristic for Steiner minimal tree problems on the Euclidean metric, *Networks* **11**, 1 (1981), 23–39.

[14] W. D. Smith, How to find Steiner minimal trees in Euclidean d-space, *Algorithmica* **7** (1992), 137–177.

[15] B. Toppur and J. M. Smith, A sausage heuristic for Steiner minimal trees in three-dimensional Euclidean space, *J. Math. Model. and Algorithms* **4** (2005), 199–217.

An Improved Bound for Orthogeodesic Point Set Embeddings of Trees

Imre Bárány* Kevin Buchin† Michael Hoffmann‡ Anita Liebenau§

Abstract

In an orthogeodesic embedding of a graph, each edge is embedded as an axis-parallel polyline that forms a shortest path in the ℓ_1 metric. In this paper we consider orthogeodesic plane embeddings of trees on grids. A grid is implicitly defined by a set $P \subset \mathbb{R}^2$ of points. Denote by Γ_P the arrangement induced by all horizontal and vertical lines that pass through a point from P. When embedding a graph on the grid defined by P, vertices are mapped to points from P and edges are realized as polylines that bend at vertices of Γ_P only. For integers n and Δ, denote by $t_\Delta(n)$ the minimum number such that for every set P of $t_\Delta(n)$ points in general position, every tree on n vertices with vertex degree at most Δ admits an orthogeodesic plane embedding on the grid defined by P. We show $t_4(n) < 11n/8$ and $t_3(n) < 9n/8$, improving an earlier bound of $3n/2$.

1 Introduction

Given a tree T on n vertices, we want to embed T on an $N \times N$ grid, for some $N \geq n$. In fact, we consider a more restricted setting where possible locations for vertices are specified in form of a set $P \subset \mathbb{R}^2$ of N points. Denote by Γ_P the arrangement induced by all horizontal and vertical lines that pass through a point from P. To embed a graph on the *grid defined by* P, vertices are mapped to points from P and edges are mapped to arcs that are polylines that bend at vertices of Γ_P only. A point set P is in *general position* if no two points have the same x- or y-coordinate.

A common theme in the study of metric graph embeddings is the desire to control the length of edges. For instance, can every edge be realized as a shortest path? In the Euclidean plane, we arrive at straight line embeddings. A natural counterpart of these embeddings on the grid is called an *orthogeodesic embedding*. In an orthogeodesic embedding, every edge is realized as an *orthogeodesic arc*, that is, a polyline that consists of axis-parallel line segments and forms a shortest path in the ℓ_1 metric. An *L-shaped* arc is an orthogeodesic arc with exactly one bend.

*Renyi Institute of Mathematics and University College London, barany@renyi.hu

†TU Eindhoven, k.a.buchin@tue.nl

‡ETH Zürich, hoffmann@inf.ethz.ch

§Monash University, Anita.Liebenau@monash.edu, work done while at FU Berlin

An embedding is *plane* if no two arcs share a common point that is not a common endpoint. Clearly an orthogeodesic plane embedding can exist only for trees of degree at most four. As it is straightforward to find orthogeodesic embeddings for paths, the only interesting cases are maximum degree three and maximum degree four. For integers n and Δ, denote by $t_\Delta(n)$ the minimum number such that for every set P of $t_\Delta(n)$ points in general position, every tree on n vertices with degree at most Δ admits an orthogeodesic plane embedding on the grid defined by P.

Di Giacomo et al. [2] showed that $t_4(n) \leq 4n-3$ and $t_3(n) \leq 3n/2$. The conference version of [2] (which is reference [1] here) claims that n points are enough for trees of degree at most three. But the proof turned out to be incomplete, as commented in the journal version. Recently, Scheucher [3] showed that $t_4(n) \leq \lfloor (3n-2)/2 \rfloor$. We improve these bounds as follows.

Theorem 1 *For every set $P \subset \mathbb{R}^2$ of $\lfloor (11n-7)/8 \rfloor$ points in general position and every tree T on $n \geq 3$ vertices of degree at most four, T admits an orthogeodesic plane embedding on the grid defined by P.*

Theorem 2 *For every set $P \subset \mathbb{R}^2$ of $\lfloor (9n-4)/8 \rfloor$ points in general position and every tree T on $n \geq 4$ vertices of degree at most three, T admits an orthogeodesic plane embedding on the grid defined by P.*

2 Proofs

For a tree T and $i \in \{1,2,3,4\}$, denote by $d_i(T)$ the number of degree i vertices in T. As a first step we prove Theorem 1 with a weaker bound of $\lfloor (3n-2)/2 \rfloor$ points and a tree T on $n \geq 2$ vertices.

Proof. The idea is to spend one extra point per vertex of degree three or four. Then we need $f(T) := |T| + d_3(T) + d_4(T)$ points. For $n \geq 2$ we have $d_1(T) = 2d_4(T) + d_3(T) + 2$. It follows that $n = |T| = \sum_{i=1}^4 d_i = 3d_4(T) + 2d_3(T) + d_2(T) + 2$ and, therefore, $f(T) = n + (n - 2 - d_2(T) - d_4(T))/2 \leq \lfloor (3n-2)/2 \rfloor$.

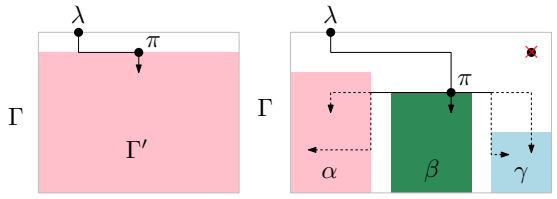
We inductively prove the following statement from which the claim follows immediately: For any tree T on $n \geq 1$ vertices, any leaf ℓ of T, any direction d of the four axis directions $\{+x, -x, +y, -y\}$, and any set Γ of $f(T)$ points in general position, there is an orthogeodesic plane embedding of T on the grid defined by Γ such that ℓ is mapped to the extreme point of Γ in direction d and every edge (no edge for

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

$n = 1$ and one edge, otherwise) connected to ℓ leaves it in the opposite direction. (For instance, if ℓ is mapped to the point λ with largest x-coordinate, then the only incident arc leaves λ on the left side.)

The statement is obviously true for $n \in \{1, 2\}$. For $n \geq 3$ we proceed as follows. Without loss of generality let $d = y$. Map ℓ to the topmost point λ of Γ (there is no choice, anyway), and consider the unique child/neighbor p of ℓ in T. Next subdivide the remaining points of Γ (other than λ) into $\deg_T(p)$ groups. We distinguish three cases depending on $\deg_T(p)$.

Case 1: If $\deg_T(p) \leq 2$, then p is a leaf of $T' = T \setminus \ell$ and we can directly apply induction to T', y, and $\Gamma \setminus \{\lambda\}$. The edge $\{\ell, p\}$ can be routed going down from λ and then turning left or right to the point π that p is mapped to (Figure 1a).



(a) $\deg_T(p) = 2$. (b) $\deg_T(p) = 4$.

Figure 1: Embedding a vertex p of degree 2 or 4.

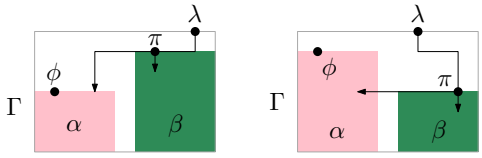
Case 2: If $\deg_T(p) = 4$, then consider the tree $T' = T \setminus \ell$. Removal of p splits T' into three components A, B and C. Obtain Γ' from Γ by removing the topmost two points. We partition Γ' into three groups: the leftmost $f(A)$ points go into a set α, the rightmost $f(C)$ points go into a set γ, and the remaining $f(T) - 2 - f(A) - f(C) = |B| + d_3(B) + d_4(B) + 1 = f(B) + 1$ points in between go into a set β. The plan is to embed A on α, $B \cup \{p\}$ on β, and C on γ.

Let π denote the topmost point of β and map p to π. We use the row of Γ below λ to route the edge between λ and π, entering λ from below (as required) and π from above. Now apply induction to three subproblems, where the vertex p and its corresponding point π are included in all of them. Note that p is a leaf in all of $A' = T' \setminus (B \cup C)$, $B' = T' \setminus (A \cup C)$, and $C' = T' \setminus (A \cup B)$, and that π is the rightmost point of $\alpha' = \alpha \cup \{\pi\}$, the topmost point of $\beta' = \beta$, and the leftmost point of $\gamma' = \gamma \cup \{\pi\}$. (Columns located between the rightmost point of α and π, which belong to β, are ignored for the purpose of handling A'. If π lies above α, also the rows between the topmost row of α and the one of π are ignored. Similarly, some columns and rows are ignored for handling C'. Essentially we always consider square grids.)

Obtain inductively an embedding for A' with p placed in direction x on α', for B' with p placed in direction y on β', and for C' with p placed in direction $-x$ on γ' (Figure 1b). The overlay of these three embeddings together with the placement of ℓ at λ and p at π forms the desired embedding for T: The only edge connected to π in α' enters π from the left side, the only edge connected to π in β' enters π from below, and the only edge connected to π in γ' enters π from the right side. As all edges within each of α', β', and γ' are orthogeodesic, no two edges from different sub-problems interfere with each other. (As the dotted lines in Figure 1b suggest, we do not know exactly how π is connected to α and γ. But we *do* know that π is accessed from one particular direction only, and so we can rely on the part shown solid, which is enough to guarantee that these edges do not interfere with those of the embedding of B'.)

Case 3: If $\deg_T(p) = 3$, then without loss of generality let the point of Γ in the row directly below λ be located to the left of λ. Consider the tree $T' = T \setminus \ell$. Removing p from T' splits the tree into two components A and B. Let $A' = T' \setminus B$ and $B' = T' \setminus A$ and partition $\Gamma' = \Gamma \setminus \{\lambda\}$ into two groups: the leftmost $f(A)$ points go into a set α and the remaining $f(T) - 2 - f(A) = |B| + d_3(B) + d_4(B) + 1 = f(B) + 1$ points go into a set β. Denote the topmost point of α by ϕ and embed p at the topmost point π of β.

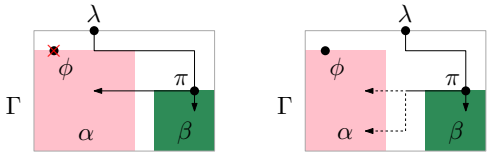
We distinguish two cases. If π is above ϕ, then by assumption π lies to the left of λ. In this case we route the edge $\{\ell, p\}$ to go down from λ and enter π from the right side (Figure 2a).



(a) π is above ϕ. (b) ϕ is above π.

Figure 2: Embedding a degree three vertex at π.

Otherwise, ϕ is above π and we route the edge $\{\ell, p\}$ to go down from λ and turn in the row of ϕ to enter π from above. This is fine, if λ lies to the right of α (Figure 2b and 3b). But if λ lies to the left of β, this routing uses part of the row of ϕ within α, which then is not available for the embedding of A' on $\alpha' = \alpha \cup \{\pi\}$. To be on the safe side, we discard ϕ from Γ (Figure 3a).

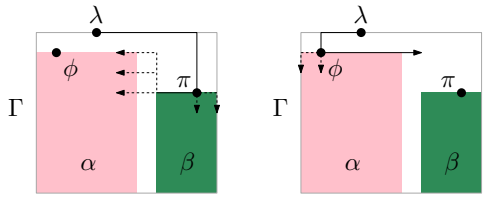


(a) λ is above α. (b) λ is between α and β.

Figure 3: If λ is above α, then ϕ is discarded.

Analogously to Case 2 we inductively obtain embeddings for A' on $\alpha \cup \{\pi\}$ and for B' on β. □

In order to improve the bound, let us consider the case of a degree three vertex p in the construction more carefully. In Case 3 above we considered three sub-cases and only in the last one (depicted in Figure 3a) a grid point needs to be discarded. If in that case the edge $\{\ell, p\}$ can be routed to leave λ on the left or right side instead of on the bottom side, then ϕ can be kept: If the right side of λ is free, then we route the edge using one bend only (Figure 4a); else if the left side of λ is free, then we move p from B to A and move the leftmost point in β to α in order to map p to ϕ instead and also route the edge $\{\ell, p\}$ using one bend only (Figure 4b).

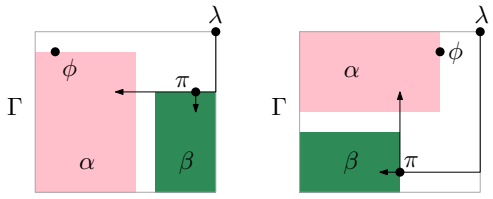


(a) right side free. (b) left side free.

Figure 4: Avoid discarding ϕ if one side of λ is free.

If ℓ has degree at most two in the original tree, then obviously at least one side of λ is free. If ℓ has degree four in the original tree, then obviously no side of λ is free. The situation is less clear in case that ℓ has degree three in the original tree.

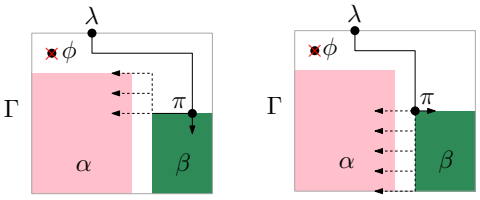
In the case depicted in Figure 2a, no side of π is free for embedding on β. But as far as α is concerned, the top side of π may be regarded as free. Although no orthogeodesic path from π to any point in α can leave π on its top side, conceptually the top side is free regardless. This point of view makes sense, because in that case λ is located at a corner of Γ and we may also regard λ as an extreme point on the (say) right side. The assignment of α and β can then be done correspondingly with respect to the right side of Γ (Figure 5) and π can be nicely accessed from the right.



(a) before (b) after

Figure 5: Switching sides if λ is at a corner.

Similarly, it can be checked that in all cases other than the one depicted in Figure 2a both α and β can access λ from at least (typically exactly) one other side. For instance, in the case depicted in Figure 3a the embedding on β can access π from the right, whereas the embedding on α can access π from above: a geodesic path cannot actually enter π from above, but it can leave α on its right side in any row above π and then move down to the row of π once it reaches the area above β to finally enter π on its left side (Figure 6a); from the perspective of α—which ignores all columns in between its right side and π—that looks like entering π from above. The path can be routed as described, unless π is the leftmost point and, hence, top-left corner of β (Figure 6b). But then we can treat π as leftmost point of β and consider the top side of π free as far as β is concerned, whereas the bottom and left side are both accessible for α.



(a) left or "above" (b) left or below

Figure 6: Possible ways for α to access π.

So during our recursive construction we discard a grid point for each degree four vertex and for certain degree three vertices. Consider a fixed processing order defined by a starting leaf of the tree. We partition the degree three vertices into two classes: For a *good* vertex v we guarantee that the left or the right side of the starting point λ is free when processing v. By the analysis above no point needs to be discarded when processing a good degree three vertex. In contrast, for a *bad* vertex no such guarantee holds. For every bad vertex v in a subtree we make an extra point available that can be discarded when processing v. Our goal is to derive a lower bound for the number of vertices that we may regard as good.

Proposition 3 *The parent of a bad degree 3 vertex is a degree 4 vertex or a good degree 3 vertex.*

Proof. Consider a bad degree 3 vertex and let p be its parent. Clearly a child of a vertex of degree at most two is good. Therefore it remains to exclude that p is a bad degree 3 vertex. When processing a bad degree 3 vertex p, we discard the point ϕ and ensure that all children of p are good (cf. Figure 6). □

Proposition 4 *If a child of a degree 3 vertex p is a bad degree 3 vertex, then the other child of v is a good degree 3 vertex.*

Proof. There is only one case where we cannot guarantee a free side at the starting point for a child of a degree three vertex: in Figure 2a, for the child of p in B to be embedded on β. If the other child of p does

not have degree three, then we select A to contain the degree three child c of p, making c good. □

Proposition 5 *If a degree 3 vertex v has a child c so that the subtree rooted at c is a path, then v is good.*

Proof. When handling v, we let A be the subtree rooted at c, to be embedded on the left point set α. The only problematic case is the one depicted in Figure 3a, where we have to show how to avoid discarding ϕ. Given that A is a path, it can be embedded on α in a monotone fashion, from right to left: every arc leaves the parent on the left and enters the child along the y-direction. In particular, the part of the row of ϕ to the right of ϕ (that is used by the arc between λ and π) is not touched. □

Propositions 3–5 allow us to classify degree three vertices during a top-down traversal of T as follows. Initially, all vertices which Proposition 5 applies to are *good*, and the remaining vertices are *unclassified*. When encountering a degree four vertex, all its unclassified degree three children are *bad*. When encountering a good degree three vertex for which both children are unclassified degree three vertices, one of the children is *bad* and the other is *good*. In all other cases, an unclassified degree three child is *good*.

Proof. (of Theorem 1) Let $T = (V, E)$ and denote $d_i = d_i(T)$. Observe that $d_1 = 2d_4 + d_3 + 2$. Let V_3^- denote the set of bad degree three vertices in T, let V_3^+ denote the set of good degree three vertices in T, and let $d_3^- = |V_3^-|$ and $d_3^+ = |V_3^+|$. Let $W \subseteq V$ denote the set of all vertices $v \in V$ such that either $v \in V_3^-$ or v is a leaf. Denote by $F \subseteq E$ the set of all edges in T that are incident to at least one vertex from W. By Propositions 3 and 5 and given $n \geq 3$, every edge in F is incident to exactly one vertex from W and one vertex from $V \setminus W$. Therefore, we can double count by the endpoints in W to obtain $|F| = 3d_3^- + d_1 = 2d_4 + 4d_3^- + d_3^+ + 2$ and by the endpoints in $V \setminus W$ to obtain $|F| \leq 4d_4 + 3d_3^+ + 2d_2$. Combining both bounds we get $2d_3^- \leq d_4 + d_3^+ + d_2 - 1$.

Setting $k = d_4 + d_3^-$, we use $n + k$ grid points for $n = 3d_4 + 2d_3 + d_2 + 2$ vertices. Therefore

$$k = \frac{3}{8}\left(n - \frac{1}{3}d_4 + \frac{2}{3}d_3^- - 2d_3^+ - d_2 - 2\right) \leq \frac{3}{8}\left(n - \frac{5}{3}d_3^+ - \frac{2}{3}d_2 - \frac{7}{3}\right) \leq \frac{3n-7}{8}.$$ □

Proof. (of Theorem 2) Define d_i, V_3^-, V_3^+, d_3^- and d_3^+ as above. If $d_3 = 0$, then T is a path and the statement is trivial. Hence suppose $d_3 \geq 1$, which implies $n \geq 4$. We consider T as a directed tree by orienting all edges away from the root. Let F denote the set of directed edges (u, v) in T such that $u \in V_3^-$ and $v \in V_3^+$. We claim that $d_3^- + |F| \leq d_3^+ - 1$.

To prove this claim, define an injective map $g\colon V_3^- \cup F \to V_3^+$. For a vertex $u \in V_3^-$ let $g(u)$ be the sibling of u in T. Such a sibling $g(u)$ exists by Proposition 3 and $g(u) \in V_3^+$ by Proposition 4. As a vertex in V_3^+ has at most one sibling, g is injective on V_3^-.

For an edge $e = (u, v) \in F$ set $g(e) = v$, where $v \in V_3^+$ by definition. As every vertex in V_3^+ has exactly one incoming edge, g is injective on F.

Note that for all vertices in $g(V_3^-)$ the parent is in V_3^+, whereas for all vertices in $g(F)$ the parent is in V_3^-. Therefore, g is injective on $V_3^- \cup F$, as claimed. It also follows that the highest vertex in V_3^+ (closest to the root) is not in $g(V_3^- \cup F)$ and, therefore, $|g(V_3^- \cup F)| \leq d_3^+ - 1$.

The claim directly generalizes to the case where F is the set of directed paths $(v_0, \ldots, v_k)$ in T such that $k \geq 1$, $v_0 \in V_3^-$, $v_k \in V_3^+$, and $\deg_T(v_i) = 2$, for $0 < i < k$. Then Propositions 3 and 5 imply $|F| = 2d_3^-$. In combination with the claim we obtain $3d_3^- \leq d_3^+ - 1$. We use $n + d_3^-$ grid points for $n = 2d_3 + d_2 + 2$ vertices. It follows that

$$d_3^- = \frac{1}{8}\left(n + 6d_3^- - 2d_3^+ - d_2 - 2\right) \leq \frac{1}{8}(n-4).$$ □

3 Conclusions

As an obvious open problem it remains to tighten the bounds for $t_3(n)$ and $t_4(n)$. Most notably, it would be nice to prove or disprove $t_3(n) = n$. No non-trivial lower bound seems to be known, even for maximum degree four and if the embedding is restricted to use L-shaped arcs only.

Acknowledgments This work was started at the 10th Gremo Workshop on Open Problems in Bergün, Switzerland, June 4–8, 2012. We thank Yoshio Okamoto and Bettina Speckmann for independently bringing the problem investigated to our attention. We also thank all participants of the workshop for the inspiring and productive atmosphere. Finally, we thank one of the reviewers for helpful comments.

References

[1] E. Di Giacomo, F. Frati, R. Fulek, L. Grilli, and M. Krug. Orthogeodesic point-set embedding of trees. *Graph Drawing (Proc. GD '11)*, volume 7034 of *Lecture Notes Comput. Sci.*, pages 52–63, 2012.

[2] E. Di Giacomo, F. Frati, R. Fulek, L. Grilli, and M. Krug. Orthogeodesic point-set embedding of trees. *Comput. Geom. Theory Appl.*, 46(8):929–944, 2013.

[3] M. Scheucher. Orthogeodesic point set embeddings of outerplanar graphs. Master's thesis, Inst. Softw. Tech., TU Graz, Graz, Austria, 2015.

Planar L-Shaped Point Set Embeddings of Trees*

Oswin Aichholzer† Thomas Hackl† Manfred Scheucher†

Abstract

In this paper we consider planar L-shaped embeddings of trees in point sets, that is, planar drawings where the vertices are mapped to a subset of the given points and where every edge consists of two axis-aligned line segments. We investigate the minimum number m, such that any n vertex tree with maximum degree 4 admits a planar L-shaped embedding in any point set of size m.

First we give an upper bound $O(n^c)$ with $c = \log_2 3 \approx 1.585$ for the general case, and thus answer the question by Di Giacomo et al. [4] whether a subquadratic upper bound exists.

Then we introduce the saturation function for trees and show that trees with low saturation can be embedded even more efficiently. In particular, we improve the upper bound for caterpillars and extend the class of trees that require only a linear number of points. In addition, we present some probabilistic results for either randomly chosen trees or randomly chosen point sets.

1 Introduction

A *point set embedding* of a given graph G in a given point set P is a drawing where all vertices are drawn as points of P. In general, the decision problem whether a graph admits a planar straight line point set embedding in a given point set is NP-complete [3], while for trees and outerplanar graphs there are efficient embedding algorithms; see for example [2]. Kaufmann and Wiese [7] have investigated a relaxation of this problem, namely point set embeddings where edges can be drawn as polylines. They proved that the decision problem remains NP-complete if at most one bend per edge is allowed, and that any planar graph admits a planar point set embedding with at most 2 bends per edge. Katz et al. [6] introduced *orthogeodesic* point set embeddings, i.e., drawings where edges have minimal L^1-length and are drawn as unions of axis parallel line segments. They proved that deciding whether a graph admits a planar orthogeodesic point set embedding is NP-complete. Di Giacomo et al. [4] introduced *L-shaped* point set embeddings, i.e., orthogeodesic point set embeddings where every edge has at most one bend, and investigated orthogeodesic and L-shaped point set embeddings of trees; in particular *caterpillars*, i.e., trees where the removal of all leaves results in a path.

As in [4, 6], throughout this paper we assume that every two points in any set of m points have distinct x- and distinct y-coordinates. Moreover, we will only consider planar L-shaped embeddings, and thus we can further assume that for every set P of m points, the x- and y-coordinates of every point in P are in $\{1, \ldots, m\}$, i.e., that $P = \{(i, \pi(i))\}_{i=1}^{m}$ holds for a permutation $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$.

For a tree T, let $f(T)$ be the minimum number m such that T admits a planar L-shaped embedding in any point set of size m, and let further $f_d(n) := \max f(T)$ where the maximum is over all trees T on n vertices with maximum degree at most d. Point sets admitting an embedding (of a certain type) of every n vertex graph (of a certain class) are referred to as *universal point sets*, see for example [4]. Obviously, only trees with maximum degree at most 4 admit planar L-shaped embeddings, and moreover, any n vertex path admits a trivial embedding in every set of n points. Therefore, only trees with maximum degree 3 or 4 are of interest. The previously best known upper bound on $f_4(n)$ is quadratic [4]. The best lower bound so far is $f_d(n) \geq n$ for $d = 3, 4$.

In Section 2, we prove $f_4(n) = O(n^{\log_2 3})$, using a recursive embedding algorithm, and hence answer the question stated by Di Giacomo et al. [4] whether a subquadratic upper bound on $f_4(n)$ exists.

In Section 3, we introduce the *saturation* function σ for trees and prove that $f(T) \leq 2^{\sigma(T)}n$ holds for any n vertex tree T. For trees with saturation bounded by a constant this clearly gives a linear upper bound, which enlarges the set of graphs that can be embedded in point sets of linear size. In particular, for caterpillars we improve the upper bound $f(T) \leq 3n - 2$ provided in [4] to $2n$, which can be further improved to $(4/3 + \varepsilon)n + O(1)$ for $\varepsilon > 0$. We further show that $f(T) = O(n^{1.5+\varepsilon})$ holds with probability at least $\frac{2\varepsilon}{1+2\varepsilon}$ if the tree T is chosen uniformly at random among all rooted n vertex trees with maximum degree at most 4.

In Section 4, we show that a given n-vertex com-

*This work is based on the Master's thesis of Manfred Scheucher [8]. Available from `http://www.ist.tugraz.at/scheucher/publ/masters_thesis_2015.pdf`. Partially supported by the ESF EUROCORES programme EuroGIGA – CRP ComPoSe, Austrian Science Fund (FWF): I648-N18 and FWF project P23629-N18 'Combinatorial Problems on Geometric Graphs'.

†Institute of Software Technology, Graz University of Technology, {oaich,thackl,mscheuch}@ist.tugraz.at

plete binary tree T can be embedded in a point set P with probability at least $1/2$, if P was chosen uniformly at random among all point sets of size $O(n \log^2 n)$. A generalization to arbitrary trees, including an improved bound on the size of the point sets, can be found in [8]. Even though the question by Di Giacomo et al. [4], whether a linear upper bound on $f_3(n)$ exists, remains open, these results give more insight into the problem.

2 A Subquadratic Bound for the General Case

For this section, we define the integer sequence $(u_n)_{n\in\mathbb{N}_0}$ recursively by

$$u_0 := 0,\ u_{n+1} := \max_{\substack{a\geq b\geq c\\ a+b+c=n}} 1 + u_a + 2u_b + 2u_c,$$

where $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ denotes the set of non-negative integers. We use a recursive approach to find a planar L-shaped embedding of a given tree T with n vertices and maximum degree at most 4 in a given point set P of size u_n. In the recursive step subtrees of sizes a, b, and c will be embedded in sub point sets of sizes u_a, u_b, and u_c, respectively. As $u_n \leq n^{\log_2 3}$ holds for every n, this will give a proof for $f_4(n) \leq n^{\log_2 3}$.

Lemma 1 *$u_n \leq n^{\log_2 3}$ holds for any $n \in \mathbb{N}_0$.*

Proof. Let $g(x) = x^{\log_2 3}$ on $[0, \infty)$. We give a proof by induction that $u_n \leq g(n)$ holds for $n \in \mathbb{N}_0$.

For the induction base, $u_0 = 0$ and $u_1 = 1$ clearly fulfill the inequality. For the induction step, let $n \geq 1$. We assume that $u_k \leq g(k)$ holds for any $k \leq n$, and prove that $u_{n+1} \leq g(n+1)$ holds. Let

$$S = \{(x, y, z) \in \mathbb{R}^3 \colon x \geq y \geq z \geq 0,\ x + y + z = n\}.$$

By definition, $u_{n+1} = 1 + u_a + 2u_b + 2u_c$ holds for some integers $a \geq b \geq c \geq 0$ with $a + b + c = n$. By the induction assumption, we can write

$$u_{n+1} \leq 1 + g(a) + 2g(b) + 2g(c),$$

and since $(a, b, c) \in S$,

$$u_{n+1} \leq \max_{(x,y,z)\in S} \underbrace{1 + g(x) + 2g(y) + 2g(z)}_{=h(x,y,z)}.$$

As g is a convex function on $[0, \infty)$, h is a convex function on S. Moreover, S is a convex set since S is spanned by $s_1 = (n, 0, 0)$, $s_2 = (\frac{n}{2}, \frac{n}{2}, 0)$, and $s_3 = (\frac{n}{3}, \frac{n}{3}, \frac{n}{3})$; a proof can be found in [8, Lemma 10].

According to the Maximum Principle, h attains its maximum over S in s_1, s_2, or s_3. We now show that $h(s_i) \leq g(n+1)$ holds for $i = 1, 2, 3$:

s_1: Due to the Mean Value Theorem it holds that $g(n+1) - g(n) = g'(\xi)$ for some $\xi \in (n, n+1)$. Since $\log_2 3 > 1$ and $1 \leq \xi$, we have $g'(\xi) = (\log_2 3)\xi^{\log_2 3 - 1} \geq 1$, and thus $h(s_1) = 1 + g(n) \leq g(n+1)$.

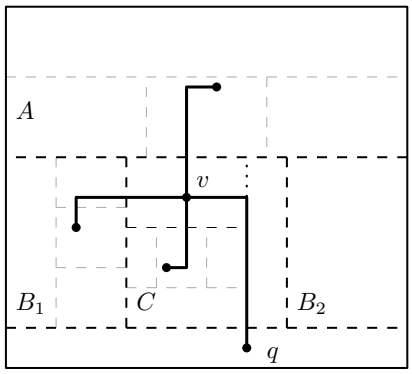


Figure 1: The recursive embedding: lines partitioning P are drawn dashed black; the dotted line illustrates how to choose C'; lines partitioning A, B, and C in the next recursion are drawn dashed gray. Only points used in the first and second recursive step are shown.

s_2: $h(s_2) = 1 + 3(n/2)^{\log_2 3} = 1 + n^{\log_2 3} = 1 + g(n) \leq g(n+1)$.

s_3: $h(s_3) = 1 + 5(n/3)^{\log_2 3} \leq 1 + g(n) \leq g(n+1)$ holds, since $5/3^{\log_2 3} < 1$.

As a consequence, we have $u_{n+1} \leq g(n+1)$. □

From the definition of u_n, we derive the following algorithm:

1. Let T be an n vertex tree with maximum degree at most 4, rooted at some degree-1-vertex r, and let P be a set of u_n points.
2. We place r at the bottommost point q of P.
3. Let v be the child of r, and let T_A, T_B, T_C be the subtrees of v of sizes a, b, and c respectively, with $a \geq b \geq c \geq 0$ and $a + b + c + 1 = n - 1$.
4. Recall that $u_n \geq u_{n-1} + 1$ holds by definition of u_n. We partition $P = A \cup (B_1 \cup C \cup B_2) \cup \{q\}$ such that A contains the topmost u_a points, q is the bottommost point, $|B_1| = |B_2| = u_b$, $|C| = 2u_c + 1$, B_1 is on the left of C, and C is on the left of B_2. Figure 1 gives an illustration.
5. At least $u_c + 1$ points in C have x-coordinates less (greater) than q. We denote this set as C'. We embed v as the topmost vertex q' in C'. As long as not all subtrees are empty, we continue recursively by embedding T_A in A, T_B in B_1 (resp., B_2), and T_C in $C' \backslash \{q'\}$. The subtrees are embedded with respect to an according rotation as illustrated in Figure 1.

Together with Lemma 1 we get the following:

Theorem 2 *$f_4(n) \leq n^{\log_2 3}$.*

For further improvements on the multiplicative factor of the $n^{\log_2 3}$ term we refer to [8, Chapter 3], where $f_3(n) \leq 0.5n^{\log_2 3} + O(n)$ and $f_4(n) \leq cn^{\log_2 3} + O(n)$ with $c \approx 0.508$ have been shown.

3 Bounds for Special Cases

In this section, we introduce the saturation function σ for trees and show how to handle trees with low saturation in a more efficient way.

Definition 1 (Saturation) *Let $T = (V, E)$ be a tree. For the rooted version T^r of T with root $r \in V$, we define $\sigma_r : V \to \mathbb{N}_0$ recursively, such that*

$$\sigma_r(v) = \max\{0, \sigma_r(u_1), \sigma_r(u_2) + 1, \ldots, \sigma_r(u_k) + 1\}$$

holds for every vertex v with children $u_1, \ldots, u_k$ ($k \geq 0$) and $\sigma_r(u_1) \geq \ldots \geq \sigma_r(u_k)$. We define the rooted saturation $\sigma(T^r) := \sigma_r(r) = \max_{v \in V} \sigma_r(v)$. *For the unrooted tree T, we define the* saturation $\sigma(T) := \min_{r \in V} \sigma(T^r)$.

Theorem 3 *$f(T) \leq 2^{\sigma(T)}n$ holds for every tree T with n vertices and maximum degree at most 4.*

Proof. We slightly modify the algorithm proposed in Section 2. For the recursion, we embed a subtree with maximum saturation in the top area instead of the largest subtree. By definition of $\sigma(T)$, $2^{\sigma(T)}$ copies of every point are sufficient for the algorithm to succeed, and thus, the statement follows. □

We remark that it is straightforward to show that $\sigma(T) \leq \log_2(n+1) - 1$ holds for every n vertex tree T (see [8, Chapter 3.3.2]), which directly gives an alternative proof for $f_4(n) = O(n^2)$.

Di Giacomo et al. [4] have already proven that, for caterpillars with maximum degree at most 4, any point set of size $3n - 2$ is sufficient to find a planar L-shaped embedding. Since caterpillars have saturation at most 1, $2n$ is an upper bound. Moreover, this upper bound can be improved to $(4/3 + \varepsilon)n + O(1)$ for any fixed $\varepsilon > 0$; we refer to [8, Chapter 5.2].

3.1 Probabilistic Analysis

In this subsection, we make use of the *register function* ρ (see e.g. Auber et al. [1]) to handle the saturation function σ: $\rho(T)$ is defined analogously to the saturation $\sigma(T)$, where the expression $\max\{0, \sigma_r(u_1), \sigma_r(u_2)+1, \sigma_r(u_3)+1, \ldots, \sigma_r(u_k)+1\}$ is changed to $\max\{0, \sigma_r(u_1), \sigma_r(u_2) + 1, \sigma_r(u_3) + 2, \ldots, \sigma_r(u_k)+k-1\}$. By definition, the register function gives an upper bound on the saturation function.

For $n \in \mathbb{N}$ let T_n denote the set of rooted trees with n vertices and maximum degree at most 4. If we suppose that every rooted tree in T_n is equally likely, then T_n can be interpreted as a random variable.

Drmota and Prodinger [5] have shown that the expected value of the random variable $\rho(T_n)$ fulfills $\mathbb{E}(\rho(T_n)) = \log_4 n + O(1)$. It is straightforward to deduce that a constant $c \in \mathbb{R}$ exists such that $\mathbb{E}(\sigma(T_n)) \leq \log_4 n + c$ holds for every n.

Theorem 4 *Let $\varepsilon > 0$. Then the probability $\mathbb{P}[f(T_n) = O(n^{1.5+\varepsilon})]$ is at least $p = \frac{2\varepsilon}{1+2\varepsilon} > 0$.*

Proof. Let $s_n = \mathbb{E}(\sigma(T_n))$. According to Markov's inequality we have $\mathbb{P}[\sigma(T_n) \geq s_n(1 + 2\varepsilon)] \leq \frac{1}{1+2\varepsilon}$, or equivalently, $\mathbb{P}[\sigma(T_n) \leq s_n(1 + 2\varepsilon)] \geq p$.

Since $s_n \leq \log_4 + c_1$ holds for a constant c_1, $2^{\sigma(T_n)} \leq 2^{s_n(1+2\varepsilon)} \leq 2^{(\log_4 n + c_1)(1+2\varepsilon)} = c_2 n^{0.5+\varepsilon}$ holds with probability at least p, where $c_2 = 2^{c_1(1+2\varepsilon)}$. The statement follows from Theorem 3. □

4 Probabilistic Approach for Trees

For a tree T let $f^{1/2}(T)$ be the minimum number m such that T admits a planar L-shaped embedding in at least half of all point sets of size m. Furthermore, let $f_d^{1/2}(n) := \max f^{1/2}(T)$ where the maximum is over all trees T on n vertices with maximum degree at most d.

4.1 Complete k-Ary Trees

We prove that $f^{1/2}(T) \leq 2(n+1)\log^2(n+1)$ holds if T is a complete binary tree on n vertices. To do so, we consider the following algorithm:

1. Let T be a complete binary tree on n vertices, rooted at r, and let P be a set of $2(n+1)\log_2^2(n+1)$ points.
2. Since T is complete, $\tilde{n} := n + 1 = 2^h$ holds with $h := \log_2 \tilde{n} \in \mathbb{N}$. We define $\alpha := 2h$ and write $|P| = \alpha\tilde{n}\log_2 \tilde{n}$.
3. We partition $P = (A \cup B_1 \cup B_2) \cup C$ such that $|A| = |C| = \alpha(\frac{\tilde{n}}{2})$, $|B_1| = |B_2| = \alpha(\frac{\tilde{n}}{2})\log_2(\frac{\tilde{n}}{2})$, $(A \cup B_1 \cup B_2)$ is above C, A is to the left of B_1, and B_1 is to the left of B_2. Furthermore, let $B = B_1 \cup B_2$. Figure 2 gives an illustration.
4. If there exists a *candidate point* q, i.e., a point in C which is to the left of B, we place r in q and continue recursively by embedding the two subtrees in B_1 and B_2, respectively. Otherwise, no solution is found and the algorithm stops.

If there exists a candidate point in every recursion (step 4), then the algorithm clearly admits a planar L-shaped embedding; one only needs to draw the edges as depicted in Figure 2 after all points have been placed.

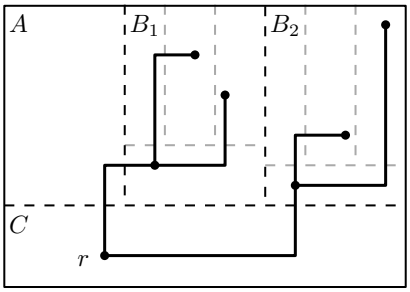


Figure 2: Embedding of a complete binary tree: black dashed lines illustrate the partition of P and gray dashed lines illustrate the partition of B_1 and B_2 in the next recursion, respectively.

It remains to show that all desired candidate points exist with probability at least 1/2.

The probability that a candidate point exists is exactly $p := 1 - \prod_{i=1}^{|C|} \left(1 - \frac{|A|+1}{|A|+|B|+i}\right)$, because assuming that neither of the points $c_1, \ldots, c_{i-1}$ is placed on the left of B, there are $|A|+|B|+i$ positions in which c_i can be placed, and $|A|+1$ of which are to the left of B. Since $i \leq |C|$ and by the partition of P, we have

$$p \geq 1 - \left(1 - \frac{|A|}{|P|}\right)^{|C|} = 1 - \left(1 - \frac{1}{2\log_2 \tilde{n}}\right)^{\alpha\frac{\tilde{n}}{2}} =: \tilde{p},$$

which we can also write as

$$\tilde{p} = 1 - \left(\left(1 - \frac{1}{2\log_2 \tilde{n}}\right)^{2\log_2 \tilde{n}}\right)^{\alpha\frac{\tilde{n}}{4\log_2 \tilde{n}}}.$$

Recall that the function $g(x) = (1 - \frac{1}{x})^x$ on $[2, \infty)$ fulfills $\frac{1}{4} \leq g(x) \leq \frac{1}{e}$ with e being Euler's number, and that the function $h(x) = \frac{x}{\ln x}$ on $(1, \infty)$ has its minimum at $x = e$ with $h(e) = e$. As a consequence $\frac{\tilde{n}}{4\log_2 \tilde{n}} \geq \frac{e\ln 2}{4} \geq \frac{\ln 2}{2}$ holds, and thus we can bound $p \geq \tilde{p} \geq 1 - (1/e)^{(\alpha \ln 2)/2} = 1 - (1/2)^{\alpha/2}$.

Obviously, this lower bound on p does not depend on the recursion level but only on α, which was chosen depending only on the initial number of points.

Since T is a complete binary tree on $n = \tilde{n} - 1$ vertices, T has $\frac{\tilde{n}}{2} - 1$ inner vertices. Furthermore, since $(1/2)^{\alpha/2} = 1/\tilde{n}$ holds by definition of α, the probability for the algorithm to succeed is at least $(1 - 1/\tilde{n})^{\tilde{n}/2-1} \geq (1 - 1/\tilde{n})^{\tilde{n}/2} \geq (1/4)^{1/2} = 1/2$.

This gives a proof of the following:

Theorem 5 *$f^{1/2}(T) \leq 2(n+1)\log^2(n+1)$ holds if T is a complete binary tree on n vertices.*

In [8, Chapter 4.1], this upper bound is improved to $O(n \log n (\log\log n)^2)$, and in [8, Chapter 4.2] $O(n \log n (\log\log n)^2)$ is shown to be an upper bound for complete ternary trees as well.

4.2 The General Case

Unfortunately, the algorithm stated in Section 4.1 can not be applied for arbitrary trees since subtrees might differ heavily in size. In [8, Chapter 4.3] a slightly modified algorithm is proposed, which makes use of a tree's Jordan center to handle this problem. Using that algorithm, they show $f_3^{1/2}(n) = O(n \log n (\log\log n)^2)$.

Even though the question by Di Giacomo et al. [4], whether a linear upper bound on $f_3(n)$ exists, remains open, we gain some more insight by this result: as any tree T admits an embedding in P with probability at least 1/2 if P was chosen uniformly at random among all point sets of size $m = O(n \log n (\log\log n)^2)$, T admits an embedding in Q with probability at least $1-(1/2)^k$ if Q was chosen uniformly at random among all point sets of size mk. Thus, we can get arbitrary close to probability 1. In particular, to get probability at least $1 - \varepsilon$ we can choose $k = \lceil \log_2(1/\varepsilon) \rceil$.

Trees with maximum degree 4 are a bit tougher to handle; while $f^{1/2}(T)$ has a quasilinear upper bound when T is a complete ternary tree, the best bound so far for the general case is $f_4^{1/2}(n) = O(n^{c+\varepsilon})$ with $c \approx 1.332$; we refer to [8, Chapter 4.3.3]. The question remains open whether a quasilinear upper bound on $f_4^{1/2}(n)$ exists.

References

[1] D. Auber, J.-P. Domenger, M. Delest, P. Duchon, and J.-M. Fédou. New strahler numbers for rooted plane trees. In *Mathematics and Computer Science III*, pages 203–215. Birkhäuser Basel, 2004.

[2] P. Bose. On embedding an outer-planar graph in a point set. *Comp. Geom. Theo. Appl.*, 23(3):303–312, 2002.

[3] S. Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *J. Graph Algorithms and Appl.*, 10(2):353–363, 2006.

[4] E. Di Giacomo, F. Frati, R. Fulek, L. Grilli, and M. Krug. Orthogeodesic point-set embedding of trees. *Comp. Geom. Theo. Appl.*, 46(8):929–944, 2013.

[5] M. Drmota and H. Prodinger. The register function for t-ary trees. *ACM Trans. on Algorithms*, 2(3):318–334, 2006.

[6] B. Katz, M. Krug, I. Rutter, and A. Wolff. Manhattan-geodesic embedding of planar graphs. In *Proc. on Graph Drawing*, pages 207–218, 2009.

[7] M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Algorithms and Appl.*, 6(1):115–129, 2002.

[8] M. Scheucher. Orthogeodesic point set embeddings of outerplanar graphs. Master's thesis, Graz University of Technology, Austria, 2015.

Drawing trees and triangulations with few geometric primitives

Gregor Hültenschmidt* Philipp Kindermann* Wouter Meulemans† André Schulz*

Abstract

We define the *visual complexity* of a plane graph drawing to be the number of geometric objects needed to represent all its edges. In particular, one object may represent multiple edges (e.g. you need only one line segment to draw two collinear edges of the same vertex). We show that trees can be drawn with $3n/4$ straight-line segments on a polynomial grid, and with $n/2$ straight-line segments on a quasi-polynomial grid. We also study the problem of drawing maximal planar graphs with circular arcs and provide an algorithm to draw such graphs using only $(5n-11)/3$ arcs. This provides a significant improvement over the lower bound of $2n$ for line segments for a nontrivial graph class.

1 Introduction

The complexity of a graph drawing can be assessed in a variety of ways (crossing number, bends, angular resolution, etc.). In this abstract, we consider the *visual complexity* of planar graphs, that is, the number of simple geometric objects necessary for any drawing. For a number of graph classes, upper and lower bounds are known for segment drawings (allowing only straight-line segments) and arc drawings (allowing circular arcs); the upper bounds are summarized in Table 1. However, these upper bounds do not require the drawings to be on the grid. A trivial lower bound is provided by $\vartheta/2$, where ϑ denotes the number of odd-degree vertices. For triangulations and general planar graphs, a lower bound of $2n + O(1)$ is known [1], where n is the number of vertices.

In this abstract, we look at segment drawings of trees on a grid and at arc drawings of triangulations. We give an algorithm that draws trees on an $O(n^2) \times O(n^{1.58})$ grid using $3n/4$ straight-line segments. This algorithm can be modified to generate drawings with an optimal $\vartheta/2$ segments on a quasi-polynomial grid; so far, no algorithms on the grid have been known. Furthermore, we prove that $(5n-11)/3$ arcs are sufficient to draw any triangulation with n vertices. We highlight that this bound is significantly lower than the $2n + O(1)$ *lower* bound known for segment drawings [2] and the so far best-known $2e/3 + O(1) = 2n + O(1)$ upper bound for circular arc drawings [4]. A straightforward extension shows that $(14n - 3e - 29)/3$ arcs are sufficient for general planar graphs with e edges.

Table 1: Upper bounds on the visual complexity for planar graphs. New results are marked with an asterisk. Here, n is the number of vertices, ϑ the number of odd-degree vertices and e the number of edges. Constant additions or subtractions have been omitted.

Class	Segment		Arc	
Trees	$\vartheta/2$	[1]	$\vartheta/2$	[1]
3-trees	$2n$	[1]	$11e/18$	[4]
3-connected	$5n/2$	[1]	$2e/3$	[4]
cubic 3-conn.	$n/2$	[3]	$n/2$	[3]
triangulation	$7n/3$	[2]	$5n/3$	*
planar	$16n/3 - e$	[2]	$14n/3 - e$	*

2 Trees with segments on the grid

Heavy paths. Let $T = (V, E)$ be an undirected tree. Our algorithm follows the basic idea of the circular arc drawing algorithm by Schulz [4]. We make use of the *heavy path decomposition* [5] of trees, which is defined as follows. First, root the tree in some vertex r. Then, for each non-leaf u, compute the size of each subtree rooted in one of its children. Let v be the child of u with the largest subtree (one of them in case of a tie). Then, (u, v) is called a *heavy edge* and all other outgoing edges of u are called *light edges*. The maximal connected components of heavy edges form the *heavy paths* of the decomposition.

We call the vertex closest to the root the *top node* of a heavy path and the subtree rooted in the top node the *heavy path subtree*. We define the *depth* of a heavy path (subtree) as follows. We treat each leaf that is not incident to a heavy edge as a heavy path of depth 0. The depth of each other heavy path is by 1 larger than the maximum depth of all heavy paths that are connected by an outgoing light edge. Heavy path subtrees of common depth are disjoint.

Boxes. We order the heavy paths nondecreasingly by their depth and then draw their subtrees in this order. Each heavy path subtree is placed completely

*FernUniversität in Hagen (Germany), gregorhueltenschmidt@gmx.de [philipp.kindermann | andre.schulz]@fernuni-hagen.de. Supported by grant SCHU 2458/4-1 (DFG).

†giCentre, City University London (United Kingdom), wouter.meulemans@city.ac.uk. Supported by Marie Skłodowska-Curie Action MSCA-H2020-IF-2014 656741.

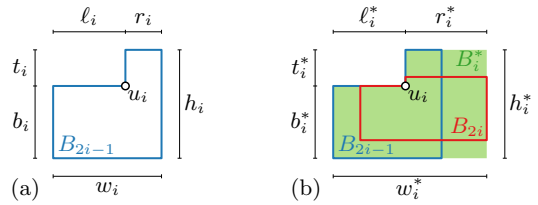


Figure 1: (a) The heavy path box B_i with top node u_i and its lengths; (b) the merged box B_i^* for B_{2i-1} and B_{2i} and its lengths.

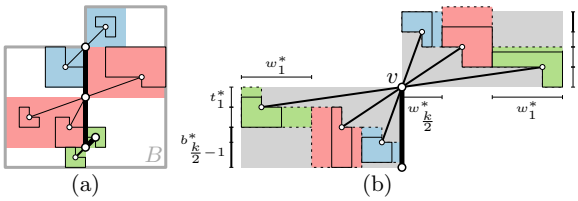


Figure 2: (a) Placement of a heavy path, its box B, and areas for the adjacent heavy path boxes. (b) Placement of the heavy path boxes adjacent to v.

inside an L-shaped box (*heavy path box*) with its top node placed at the reflex angle; see Fig. 1a for an illustration of a heavy path box B_i with top node u_i, width $w_i = \ell_i + r_i$, and height $h_i = t_i + b_i$. We require that (i) heavy path boxes of common depth are disjoint, (ii) u_i is the only vertex on the boundary, and (iii) $b_i \geq t_i$. Note that the boxes will be mirrored horizontally and/or vertically in some steps of the algorithm. We draw each heavy path subtree of depth 0 as a heavy path box B_i with $\ell_i = r_i = t_i = b_i = 1$.

Drawing. Assume that we have already drawn each heavy path subtree of depth k. When drawing the subtree of a heavy path $\langle v_1, \ldots, v_m \rangle$ of depth $k+1$, we proceed as follows. The last vertex on a heavy path has to be a leaf, so v_m is a leaf. If $\text{outdeg}(v_{m-1})$ is odd, we place the vertices $v_1, \ldots, v_m$ on a vertical line; otherwise, we place only the vertices $v_1, \ldots, v_{m-1}$ on a vertical line and treat v_m as a heavy path subtree of depth 0 that is connected to v_{m-1}. For $1 \leq h \leq m-1$, all heavy path boxes adjacent to v_h will be drawn either in a rectangle on the left side of the edge (v_h, v_{h+1}) or in a rectangle on the right side of the edge (v_{h-1}, v_h) (a rectangle that has v_1 as its bottom left corner for $h = 1$); see Fig. 2a for an illustration with even $\text{outdeg}(v_{m-1})$.

We now describe how to place the heavy path boxes $B_1, \ldots, B_k$ with top node $u_1, \ldots, u_k$, respectively, incident to some vertex v on a heavy path into the rectangles described above. First, assume that k is even. Then, for $1 \leq i \leq k/2$, we order the boxes such that $b_{2i} \leq b_{2i-1}$. We place the box B_{2i-1} in the lower left rectangle and box B_{2i} in the upper right rectangle in such a way that the edges (v, u_{2i-1}) and (v, u_{2i}) can be drawn with a single segment. To this end, we construct a *merged* box B_i^* as depicted in Fig. 1b with $\ell_i^* = \max\{\ell_{2i-1}, \ell_{2i}\}$, $r_i^* = \max\{r_{2i-1}, r_{2i}\}$, and $w_i^* = \ell_i^* + r_i^*$; the heights are defined analogously. We mirror all merged boxes horizontally and place them in the lower left rectangle (of width $\sum_{i=i}^{k/2} w_i^*$) as follows. We place B_1^* in the top left corner of the rectangle. For $2 \leq j \leq k/2$, we place B_j^* directly to the right of B_{j-1}^* such that its top border lies exactly t_{j-1}^* rows below the top border of B_{j-1}^*. Symmetrically, we place the merged boxes (vertically mirrored) in the upper right rectangle. Finally, we place each box B_{2i-1} (horizontally mirrored) in the lower left copy of B_i^* such that their inner concave angles coincide, and we place each box B_{2i} (vertically mirrored) in the upper right copy of B_i^* such that their inner concave angles coincide; see Fig. 2b. If k is odd, we simply add a dummy box $B_{k+1} = B_k$ that we remove afterwards.

Analysis. We will now calculate the width w_v and the height h_v of this construction. For the width, we have

$$w_v = 2\sum_{i=1}^{k/2} w_i^* = 2\sum_{i=1}^{k/2} \max\{w_{2i-1}, w_{2i}\} \leq 2\sum_{i=1}^{k} w_i.$$

The height of each rectangle in the construction is at least $2\sum_{i=1}^{k/2} t_i^*$, but we have to add a bit more for the bottom parts of the boxes; in the worst case, this is $\max_{1\leq j\leq k/2} b_{2j-1}$ in the lower rectangle and $\max_{1\leq h\leq k/2} b_{2h}$ in the upper rectangle. Since we require $b_i \geq t_i$ for each i, we have

$$\begin{aligned} h_v &\leq 2\sum_{i=1}^{k/2} t_i^* + \max_{1\leq j\leq k/2} b_{2j-1} + \max_{1\leq h\leq k/2} b_{2h} \\ &\leq 2\sum_{i=1}^{k} t_i + \sum_{j=1}^{k} b_i \leq \frac{3}{2}\sum_{i=1}^{k} h_i. \end{aligned}$$

Since all heavy path trees of common depth are disjoint, the heavy path boxes of common depth are also disjoint. Further, we place only the top vertex of a heavy path on the boundary of its box. Finally, since we order the boxes such that $b_{2i} \leq b_{2i-1}$ for each i, for the constructed box B we have $b \geq t$.

Due to the properties of a heavy path decomposition, the maximum depth is $\lceil \log n \rceil$. Recall that we place the depth-0 heavy paths in a box of width and height 2. Hence, by induction, a heavy path subtree of depth d with n' vertices lies inside a box of width $2 \cdot 2^d \cdot n'$ and height $2 \cdot (3/2)^d \cdot n'$. Thus, the whole tree is drawn in a box of width $2 \cdot 2^{\lceil \log n \rceil} n = O(n^2)$ and height $2 \cdot (3/2)^{\lceil \log n \rceil} n = O(n^{1+\log 3/2}) \subseteq O(n^{1.58})$. Following the analysis of Schulz [4], the drawing uses at most $\lceil 3e/4 \rceil = \lceil 3(n-1)/4 \rceil$ segments.

Theorem 1 *Every tree admits a straight-line drawing that uses at most $\lceil 3e/4 \rceil$ arcs on an $O(n^2) \times O(n^{1.58})$ grid.*

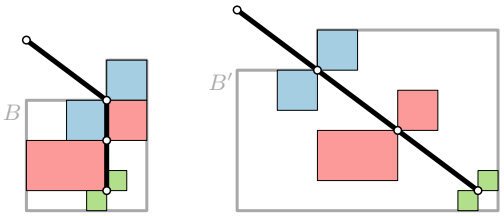


Figure 3: Further improvement on the visual complexity via increasing the size of heavy path boxes. The old box B and the modified box B'.

We finish this section with an idea of how to get a grid drawing with the best possible number of straight-line segments. Due to the limited space we give only a sketch. Observe that there is only one situation in which the previous algorithm uses more segments than necessary, that is the top node of each heavy path. This suboptimality can be "repaired" by *tilting* the heavy path as sketched in Fig. 3. Note that the incident subtrees with smaller depth will only be translated. To make this idea work, we have to blow up the size of the heavy path boxes. We are left with scaling in each "round" by a polynomial factor. Since there are only $\log n$ rounds, we obtain a drawing on a quasi-polynomial grid.

Theorem 2 *Every tree admits a straight-line drawing with the smallest number of straight-line segments on a quasi-polynomial grid.*

3 Triangulations with circular arcs

As used in previous articles [2, 4], a canonical order $v_1, \ldots, v_n$ on the vertices of a triangulation structures our drawing algorithm. However, we use the order in reverse. We start by drawing v_1, v_2, and v_n on a circle; see Fig. 4a. We assume that they are placed as shown and hence refer to the arc connecting v_1 and v_2 as *the bottom arc*. The interior of the circle is the *undrawn* region $\mathcal{U}$ which we maintain as a strictly convex shape. The vertices incident to $\mathcal{U}$ are referred to as the *horizon* and denoted in order, $h_1, h_2, \ldots, h_{k-1}, h_k$; we maintain that $h_1 = v_1$ and $h_k = v_2$. Initially, we have $k = 3$ and $h_2 = v_n$. We iteratively take a vertex h_i of the horizon (the latest in the canonical order) to process it. Processing a vertex means that we draw its undrawn neighbors and edges between these, thereby removing h_i from the horizon.

Invariant. We maintain as invariant that each vertex v (except v_1, v_2, and v_n) has a segment ℓ_v incident from above such that its downward extension intersects the bottom arc strictly between v_1 and v_2. Observe that, since $\mathcal{U}$ is strictly convex, this and h are the only intersection points for ℓ_h with the undrawn region's boundary for a vertex h on the horizon.

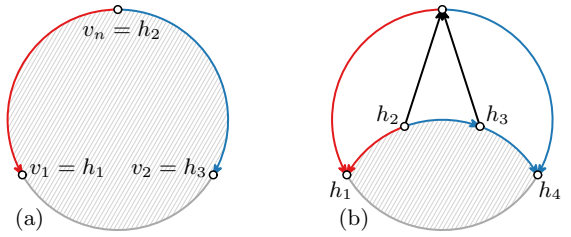


Figure 4: (a) Initial state of the algorithm. (b) State of the algorithm after processing v_n. Hatching indicates undrawn region.

Processing a vertex. To process a vertex h_i, we first consider the triangle $h_{i-1}h_ih_{i+1}$: this triangle (except for its corners) is strictly contained in $\mathcal{U}$. We draw a circular arc A from h_{i-1} to h_{i+1} with maximal curvature, but within this triangle; see Fig. 5a. This ensures a plane drawing, maintaining a strictly convex undrawn region. Moreover, it ensures that h_i can "see" the entire arc A.

Vertex h_i may have a number of neighbors that were not yet drawn. To place these neighbors, we dedicate a fraction of the arc A. In particular, this fraction is determined by the intersections of segments v_1h_i and v_2h_i with A; see Fig. 5b. By convexity of $\mathcal{U}$, these intersections exist. If h_{i-1} is equal to v_1, the intersection for v_1h_i degenerates to v_1; similarly, the intersection of v_2h_i may degenerate to v_2. We place the neighbors in order along this designated part of A, drawing the relevant edges as line segments. This implies that all these neighbors obtain a line segment that extends to intersect the bottom arc, maintaining the invariant. We position one neighbor to be a continuation of segment ℓ_{h_i}, which by the invariant must extend to intersect the designated part of A as well.

Schnyder woods. Using a Schnyder realizer of the triangulation, we decompose the edges into three trees: T_1, T_2, and T_n rooted at v_1, v_2, and v_n, respectively. Following the rationale of Durocher and Mondal [2], we assume w.l.o.g. that T_n has the smallest number of leaves. In particular, the total number of leaves in a minimal realizer is upper bounded by $2n - 5$ [2]. Hence, T_n has at most $(2n - 5)/3$ leaves.

Complexity. We start with one circle and subsequently process $v_n, \ldots, v_4$, adding one circular arc per vertex (representing edges in T_1 and T_2) and a number of line segments (representing edges in T_n). Note that processing v_3 has no effect since the edge v_1v_2 is the bottom arc. Counting the circle as one arc, we thus have $n - 2$ arcs in total. At every vertex in T_n, one incoming edge is collinear with the outgoing one towards the root. We charge each line segment to a leaf of T_n: there are at most $(2n - 5)/3$ segments.

Thus, the total visual complexity is at most $n - 2 + (2n - 5)/3 = (5n - 11)/3$. In particular, this shows

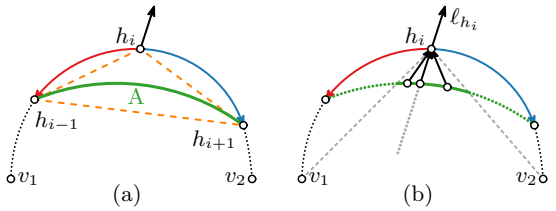


Figure 5: (a) Arc A lies inside the dashed triangle $h_{i-1}h_ih_{i+1}$. (b) Undrawn neighbors of h_i are placed on A, in the section determined by v_1 and v_2. One neighbor is placed to align with ℓ_{h_i} towards a predecessor of h_i.

that, with circular arcs, we obtain greater expressive power for a nontrivial class of graphs in comparison to the $2n$ lower bound that is known for drawing triangulations with line segments.

Degrees of freedom. One circular arc has five degrees of freedom (DoF) which is one more than a line segment. In this light, our algorithm with circular arcs uses roughly $5 \cdot 5n/3 = 25n/3$ DoF. We disregard any DoF reduction arising from the need to have arcs coincide at vertices. This remains an improvement over the result of Durocher and Mondal [2], using roughly $4 \cdot 7n/3 = 28n/3$ DoF. The lower bound for line segments ($4 \cdot 2n = 24n/3$) is lower than what we seem to achieve with our algorithm. However, our algorithm uses line segments rather than arcs to draw the tree T_n. Thus, the actual DoF employed by the algorithm is roughly $5n + 4 \cdot 2n/3 = 23n/3$, which is in fact below the lower bound for line segments.

Theorem 3 *Every triangulation admits a circular arc drawing that uses at most* $(5n - 11)/3$ *arcs.*

4 General planar graphs

Simple bound. The algorithm for triangulations easily adapts to draw a general planar graph G with n vertices and e edges. As connected components can be drawn independently, we assume G is connected. We need to only triangulate G, thereby adding $3n - e - 6$ chords. We then run the algorithm described in the previous section, using $(5n - 11)/3$ arcs. Finally, we remove the chords from the drawing. Each chord may split an arc into two arcs, thereby increasing the total complexity by one. We obtain a drawing of G using $(5n - 11)/3 + 3n - e - 6 = (14n - 3e - 29)/3$ arcs.

Improved bound. We may do slightly better by finding a "good" triangulation, using the property that at most two arcs at every vertex continue: one for the horizon and one for T_n. We reduce the necessary geometric primitives by picking a single vertex on every face and connecting all chords to that particular vertex. (We may even further save on complexity by selecting the same vertex for two adjacent faces.) This saves us an additional $\max\{0, |f| - 5\}$ on complexity for a face f of size $|f|$: it needs $|f| - 3$ chords, but only two of these can increase the complexity on removal. We can thus obtain a complexity of $(14n - 3e - 29)/3 - \sum_{f \in G} \max\{0, |f| - 5\}$. In other words, this approach reduces the complexity upper bound by $R = \sum_{f \in G} \max\{0, |f| - 5\}$. Below, we show that $R \geq \max\{0, 5n - 3e\}$, thus implying an overall upper bound of $(14n - 3e - 29)/3 - \max\{0, 5n - 3e\}$.

If $3e \geq 5n$, all faces may have size 5 and thus $R = 0$ is possible. We find that $(14n - 3e - 29)/3 \leq (14n - 5n - 29)/3 = 3n - 29/3$.

Since sparsity increases the upper bound, we construct a worst-case sparse graph to determine the lowest value of R. Consider an arbitrary connected graph G on n vertices with e edges and consider the sizes of all faces. If there there are two faces f' and f'' with $|f''| < 5 < |f'|$, we can reduce R by one by "reassigning" one chord of f' to f''. We ignore here whether the new graph with the given face sizes can actually be realized. We can also reassign a chord from f' to f'' if $|f''| \geq |f'| > 5$ without effecting R. Hence, a worst-case can be obtained if all faces but one have size at most 5; let f denote this one other face. This effectively reduces R to $\max\{0, |f| - 5\}$.

Double counting of edges along faces gives us $2e = \sum_{f' \in G} |f'| \leq |f| + 5(F - 1)$, where $F = 2 + e - n$ is the total number of faces in G. Hence, we find that $|f| \geq 5n - 3e + 5$. For $R = \max\{0, |f| - 5\}$ to be equal to $|f| - 5$, we need that $|f| \geq 5$, which is implied by $3e \leq 5n$. If this is indeed the case, then R is equal to $5n - 3e$. The upper bound is then $(14n - 3e - 29)/3 - (5n - 3e) = 2e - n/3 - 29/3$. Using $3e < 5n$ we find that this is at most $2(5n/3) - n/3 - 29/3 = 3n - 29/3$.

Theorem 4 *Every planar graph admits a circular arc drawing with at most* $\min\{3n, 14n/3 - e\} - 29/3$ *arcs.*

References

[1] V. Dujmovic, D. Eppstein, M. Suderman, and D. R. Wood. Drawings of planar graphs with few slopes and segments. *Comp. Geom. Theory & Appl.*, 38(3):194–212, 2007.

[2] S. Durocher and D. Mondal. Drawing plane triangulations with few segments. In *Proc. 26th Can. Conf. Comp. Geom. (CCCG'14)*, pages 40–45, 2014.

[3] A. Igamberdiev, W. Meulemans, and A. Schulz. Drawing planar cubic 3-connected graphs with few segments: Algorithms and experiments. In *Proc. 23rd Int. Symp. Graph Drawing & Netw. Vis. (GD'15)*, LNCS 9411, pages 113–124, 2015.

[4] A. Schulz. Drawing graphs with few arcs. *J. Graph Alg. & Appl.*, 19(1):393–412, 2015.

[5] R. E. Tarjan. Linking and cutting trees. In *Data Struct. & Netw. Alg.*, pages 59–70. SIAM, 1983.

Strongly Monotone Drawings of Planar Graphs*

Stefan Felsner† Alexander Igamberdiev‡ Philipp Kindermann† Boris Klemz§
Tamara Mchedlidze¶ Manfred Scheucher‖

Abstract

A straight-line drawing of a graph is called *monotone* if for each pair of vertices there exists a path which is monotonically increasing in some direction, and it is called a *strongly monotone* if the direction of monotonicity is given by the direction of the line segment connecting the two vertices.

We present algorithms to compute crossing-free strongly monotone drawings for some classes of planar graphs; namely, 3-connected planar graphs, outerplanar graphs, and 2-trees. The drawings of 3-connected planar graphs are based on primal-dual circle packings. Our drawings of outerplanar graphs depend on a new algorithm that constructs strongly monotone drawings of trees which are also convex. For trees without degree-2 vertices, these drawings are strictly convex.

1 Introduction

When reading data visualized as a drawing of a graph, a common task is to find a path between a source vertex and a target vertex. This task serves as the motivation for the following quality criterion for graph drawings.

Let Γ be a straight-line drawing of graph $G = (V, E)$. We say that a path P in Γ is *monotone with respect to* a direction (or vector) d if the orthogonal projections of the vertices of P on a line with direction d appear in the same order as in P. Drawing Γ is called *monotone* if for each pair of vertices $u, v \in V$ there is a connecting path that is monotone with respect to some direction. To support the path-finding tasks it is useful to restrict the monotone direction for each path to the direction of the line segment connecting the source and the target vertex: a path $v_1 v_2 \dots v_k$ is called *strongly monotone* if it is monotone with respect to the vector $\overrightarrow{v_1 v_k}$. Drawing Γ is called *strongly monotone* if each pair of vertices $u, v \in V$ is connected by a strongly monotone path. We are interested in strongly monotone drawings which are also planar. If crossings are allowed, then any strongly monotone drawing of a spanning tree of G yields a strongly monotone drawing of G [1]. For the results stated in this abstract we interpret monotonicity in a strict sense, i.e., we do not allow edges on the path that are orthogonal to the segment between the endpoints.

Is has been shown that every connected planar graph admits a monotone drawing on a grid of size $O(n) \times O(n^2)$ [4]. On the other hand, there exists an infinite class of 1-connected graphs that do not admit strongly monotone drawings [5]. Any tree and any 2-connected outerplanar graph has a strongly monotone drawing [5]. It is known that the area required for strongly monotone drawings of trees and binary cacti is exponential [6].

In this work, we show that any 3-connected planar graph admits a strongly monotone drawing (Section 2). Then, we answer in the affirmative the open question of Kindermann et al. [5] on whether every tree has a strongly monotone drawing which is (strictly) convex. We use this result to show that every outerplanar graph admits a strongly monotone drawing (Section 3). Finally, we prove that 2-trees can be drawn strongly monotone (Section 4). All our proofs are constructive and admit efficient drawing algorithms. Our main open question is whether every planar 2-connected graph admits a plane strongly monotone drawing. Due to space constraints we either sketch or omit the proofs; detailed proofs can be found in the full preprint version [3].

2 3-Connected Planar Graphs

In this section, we prove the following:

Theorem 1 *Every 3-connected planar graph has a strongly monotone drawing.*

*This research was initiated during the Geometric Graphs Workshop Week (GGWeek'15) at the FU Berlin in September 2015. Work by P. Kindermann was supported by DFG grant SC2458/4-1. Work by M. Scheucher was partially supported by the ESF EUROCORES programme EuroGIGA – CRP ComPoSe, Austrian Science Fund (FWF): I648-N18 and FWF project P23629-N18 'Combinatorial Problems on Geometric Graphs'.

†Institut für Mathematik, Technische Universität Berlin, Germany
‡LG Theoretische Informatik, FernUniversität in Hagen, Germany
§Institute of Computer Science, Freie Universität Berlin, Germany
¶Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
‖Institute of Software Technology, Graz University of Technology, Austria

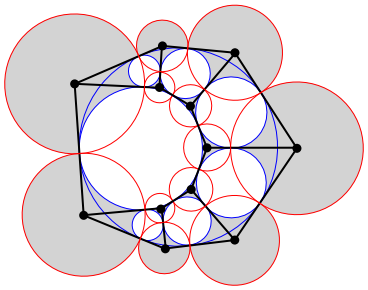


Figure 1: A primal-dual circle packing. Vertex circles in red, face circles in blue, regions of faces in white and regions of vertices in gray.

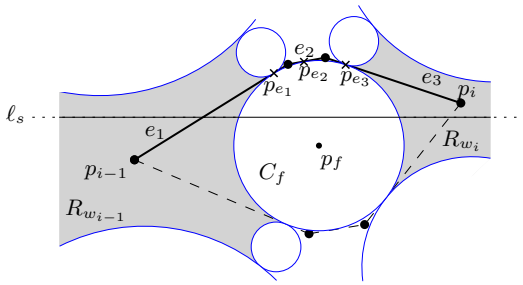


Figure 2: The path P_i connecting p_{i-1} and p_i.

We show that the straight-line drawing corresponding to a primal-dual circle packing of a graph G is strongly monotone. The theorem then follows from the fact that any 3-connected planar graph $G = (V, E)$ admits a primal-dual circle packing [2].

A *primal-dual circle packing* of a plane graph G consists of two families $\mathcal{C}_V$ and $\mathcal{C}_F$ of circles such that, there is a bijection $v \leftrightarrow C_v$ between the set V of vertices of G and circles of $\mathcal{C}_V$ and a bijection $f \leftrightarrow C_f$ between the set F of faces of G and circles of $\mathcal{C}_F$. Moreover, the following three properties hold: (**I**) The circles in the family $\mathcal{C}_V$ are interiorly disjoint and their contact graph is G, i.e., $C_u \cap C_v \neq \emptyset$ if and only if $(u, v) \in E(G)$. (**II**) If $C_o \in \mathcal{C}_F$ is the circle of the outer face o, then the circles of $\mathcal{C}_F \setminus \{C_o\}$ are interiorly disjoint while C_o contains all of them. The contact graph of $\mathcal{C}_F$ is the dual G^* of G, i.e., $C_f \cap C_g \neq \emptyset$ if and only if $(f, g) \in E(G^*)$. (**III**) The circle packings C_V and C_F are orthogonal, i.e., if $e = (u, v)$ and the dual of e is $e^* = (f, g)$, then there is a point $p_e = C_u \cap C_v = C_f \cap C_g$; moreover, the common tangents t_{e^*} of C_u, C_v and t_e of C_f, C_g cross perpendicularly in p_e.

Let a primal-dual circle packing of a graph G be given. For each vertex v, let p_v be the center of the corresponding circle C_v. By placing each vertex v at p_v, we obtain a planar straight-line drawing Γ of G. In this drawing, the edge $e = (u, v)$ is represented by the segment with end-points p_u and p_v on t_e. The face circles are inscribed circles of the faces of Γ; moreover, C_f is touching each boundary edge of the face f; see Figure 1. A straight-line drawing Γ^* of the dual G^* of G with the dual vertex of the outer face o at infinity can be obtained similarly by placing the dual vertex of each bounded face f at the center of the corresponding circle C_f. In this drawing, a dual edge $e^* = (f, o)$ is represented by the ray supported by t_{e^*} that starts at p_f and contains p_e.

We make use of a specific partition Π of the plane; Figure 1 gives an illustration. The regions of Π correspond to the vertices and the faces of G. For a vertex or face x, let D_x be the interior of the disk C_x. We define the region R_f of a bounded face f as D_f. The region R_v of a vertex v is obtained from the disk D_v by removing the intersections with the disks of bounded faces, i.e., $R_v = D_v \setminus \bigcup_{f \neq o} R_f = D_v \setminus \bigcup_{f \neq o} D_f$. To get a partition of the whole plane, we assign the complement of the already defined regions to the outer face. Note that the edge-points p_e are part of the boundary of four regions of Π. Additionally, if two regions of Π share more than one point on the boundary, then one of them is a vertex region R_v, the other is a face-region D_f, and v is incident to f in G.

We are now prepared to prove the strong monotonicity of Γ. Consider two vertices u and v and let ℓ be the line spanned by p_u and p_v. W.l.o.g., assume that ℓ is horizontal and p_u lies left of p_v. Let ℓ_s be the directed segment from p_u to p_v. Since $p_u \in R_u$ and $p_v \in R_v$, the segment ℓ_s starts and ends in these regions. In between, the segment will traverse some other regions of Π. This is true unless (u, v) is an edge of G whence the strong monotonicity for the pair is trivial. We assume *non-degeneracy* in the sense that the interior of the segment ℓ_s contains no vertex-point p_w, edge-point p_e, or face-point p_f.

Let $u = w_0, w_1, \ldots, w_k = v$ be the sequence of vertices whose region is intersected by ℓ_s, in the order of intersection from left to right and let $p_i = p_{w_i}$. We will construct a strongly monotone path P from p_u to p_v in Γ that contains $p_u = p_0, p_1, \ldots, p_k = p_v$ in this order. We show how to construct P_i, the subpath of P from p_{i-1} to p_i. Since ℓ_s may revisit a vertex-region, it is possible that $p_{i-1} = p_i$; in this case we set $P_i = p_i$. Now suppose that $p_{i-1} \neq p_i$. Non-degeneracy implies that the segment ℓ_s alternates between vertex-regions and face-regions; hence, a unique disk D_f is intersected by ℓ_s between the regions of w_{i-1} and w_i. It follows that w_{i-1} and w_i are vertices on the boundary of f. The boundary of f contains two paths from w_{i-1} to w_i. In Γ, one of these two paths from p_{i-1} to p_i is above D_f; we call it the *upper path*, the other one is below D_f, this is the *lower path*. If the center p_f of D_f lies below ℓ, we choose the upper path from p_{i-1} to p_i as P_i; otherwise, we choose the lower path.

Suppose that this rule led to the choice of the upper path; see Figure 2. The case that the lower

path was chosen works analogously. We have to show that P_i is monotone with respect to ℓ_s, i.e., to the x-axis. Let $e_1, \ldots, e_r$ be the edges of this path and let $e_j = (q_{j-1}, q_j)$; in particular $q_0 = p_{i-1}$ and $q_r = p_i$. Since $R_{w_{i-1}}$ is star-shaped with center p_{i-1}, the segment connecting p_{i-1} with the first intersection point of ℓ with C_f belongs to $R_{w_{i-1}}$. Therefore, the point p_{e_1} of tangency of edge e_1 at C_f lies above ℓ. Similarly, p_{e_r} and, hence, all the points p_{e_j} lie above ℓ. Since the points $p_{e_1}, \ldots, p_{e_r}$ appear in this order on C_f and the center of C_f lies below ℓ, we obtain that their x-coordinates are increasing in this order. This sequence is interleaved with the x-coordinates of $q_0, q_1, \ldots, q_r$, whence this is also monotone. This proves that the chosen path P_i is monotone with respect to ℓ. Monotonicity also holds for the concatenation $P = P_1 + P_2 + \ldots + P_k$.

Even if degenerancy is allowed, there still exists a strongly monotone path consisting of the edges tangent to the circles intersected by l_s. This can be shown by carefully examining the arising special cases.

3 Trees and Outerplanar Graphs

Consider a straight-line, crossing-free drawing Γ of a tree and replace each edge that leads to a leaf by a ray that begins with the edge and extends accross the leaf. If all the unbounded polygonal regions in the obtained drawing Γ' are convex, then drawing Γ is called *convex*. If all angles in Γ' are less than π, then Γ is called *strictly* convex.

Kindermann et al. [5] have shown that any tree has a strongly monotone drawing and that any binary tree has a strictly convex strongly monotone drawing. They left as an open question whether every tree admits a convex strongly monotone drawing; noticing that, in the positive case, this would imply that every Halin graph has a convex strongly monotone drawing. We give an affirmative answer to this question by stating the following:

Theorem 2 *Every tree has a convex strongly monotone drawing. If the tree has no degree-2 vertex, then the drawing is strictly convex.*

The theorem can be proven by inductively generating a corresponding drawing. In the beginning some root vertex v_0 is placed in the plane and its k children are placed at the corners of a regular k-gon with center v_0. The drawing if then generated by iteratively expanding leafs while maintaining the following:

Invariant: (**I**) Every leaf is located on a corner of the convex hull of the vertices. (**II**) If $a_1, \ldots, a_\ell$ is the counterclockwise order of the leaves on the convex hull, then for $i = 1, \ldots, \ell$ the vectors $(\overrightarrow{a_i a_{i-1}})^\perp$, $\overrightarrow{p_i, a_i}$, $(\overrightarrow{a_{i+1} a_i})^\perp$ appear in counterclockwise radial order, where p_i denotes the unique vertex adjacent to a_i. (**III**) The angle between two consecutive edges incident to a vertex v is at most π and is equal to π only when v has degree two. (**IV**) Γ is strongly monotone.

We can utilize Theorem 2 to show that every outerplanar graph has a strongly monotone drawing that is *convex*, i.e. every internal face is realized as a convex region.

Theorem 3 *Every outerplanar graph has a convex strongly monotone drawing.*

Proof. Let G be an outerplanar graph with at least 2 vertices. For every vertex $v \in V$, we add two dummy vertices v', v'' and edges (v, v'), (v, v''). By construction, the resulting graph H is outerplanar and does not contain vertices of degree 2. Let Γ_H be an outerplanar drawing of H. We will construct a convex strongly monotone drawing Γ'_H of H with the same combinatorial embedding as Γ_H.

Let T be an arbitrary spanning tree of H. By construction, no vertex in T has degree 2. Thus, according to Theorem 2, T admits a strongly monotone drawing Γ_T which is strictly convex and which also preserves the order of the children for every vertex, i.e., the rotation system coincides with the one in Γ_H.

Now, we insert all the missing edges. Recall that, by removing an edge from a planar drawing, the two adjacent faces are merged. Since the drawing Γ_T of T is strictly convex and since Γ_T preserves the rotation system of Γ_H, by inserting an edge e of the graph H into Γ_T one strictly convex face is partitioned into two strictly convex faces. Furthermore, the insertion of an edge does not destroy strong monotonicity. We reinsert all edges of H iteratively. The resulting drawing Γ'_H of H is a strictly convex and strongly monotone.

Finally, we remove all the dummy vertices and obtain a strongly monotone drawing of G. Since Γ'_H has the same combinatorial embedding as Γ_H, every dummy vertex lies in the outer face. Hence, no internal face is affected by the removal of dummy vertices, and thus all interior faces remain strictly convex. □

4 2-Trees

A *2-tree* is a graph produced by starting with a K_3 and then repeatedly adding vertices such that each added vertex v has exactly two neighbours v_1, v_2 and there is an edge $e = (v_1, v_2)$. We say that v is *stacked* on e. In this section, we provide a proof sketch for the following theorem:

Theorem 4 *Every 2-tree admits a strongly monotone drawing.*

We begin by introducing some notation. A *drawing with bubbles* of a graph $G = (V, E)$ is a straight-line drawing of G in the plane such that, for some $E' \subseteq E$, every edge $e \in E'$ is associated with a circular region

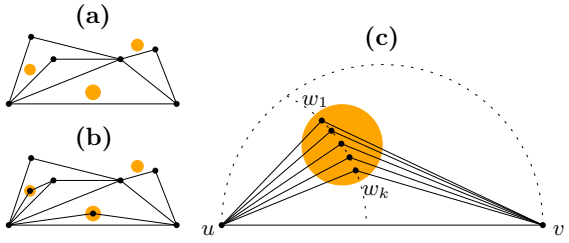


Figure 3: A drawing with bubbles (a) together with an extension (b). Stacking vertices into a bubble (c).

in the plane, called a *bubble* B_e; see Figure 3(a). An *extension* of a drawing with bubbles is a straight-line drawing that is obtained by taking some subset of edges with bubbles $E'' \subseteq E'$ and stacking one vertex on top of each edge $e \in E''$ into the corresponding bubble B_e; see Figure 3(b). (Since every bubble is associated with a unique edge we often simply say that a vertex is stacked into a bubble without mentioning the corresponding edge.) We call a drawing with bubbles Γ strongly monotone if *every* extension of Γ is strongly monotone. Note that this implies that if a vertex w is stacked on top of edge e into bubble B_e, then there exists a strongly monotone path from w to any other vertex in the drawing and, furthermore, there exists a strongly monotone path from w to any of the current bubbles, i.e., to any vertex that might be stacked into another bubble.

Every 2-tree $T = (V, E)$ can be constructed through the following iterative procedure: **1.** Start with one edge and tag it as *active*. During the entire procedure, every present edge is tagged either as active or *inactive*. **2.** Pick one active edge e and stack vertices $w_1, \ldots, w_k$ on top of this edge for some $k \geq 0$ (we note that k might equal 0). Edge e is then tagged as inactive and all new edges incident to the stacked vertices $w_1, \ldots, w_k$ are tagged as active. **3.** If there are active edges remaining, repeat Step 2.

Observe that Step 2 is performed exactly once per edge and that an according decomposition for T can always be found by the definition of 2-trees. We construct a strongly monotone drawing of T by geometrically implementing the iterative procedure described above, so that after every step of the algorithm the present part of the graph is realized as a drawing with bubbles. We maintain the following:

Invariant: After each step of the algorithm every active edge comes with a bubble and the drawing with bubbles is strongly monotone. Additionaly, for an edge $e = (u, v)$ with bubble B_e for each point $w \in B_e$, the angle $\angle(\overrightarrow{uw}, \overrightarrow{wv})$ is obtuse.

In Step 1, we arbitrarily draw the edge e_0 in the plane. Clearly, it is possible to define a bubble for e_0 that only allows obtuse angles. In Step 2, we place the vertices $w_1, \ldots, w_k$ over an edge $e = (u, v)$ as follows. The fact that stacking a vertex into B_e gives an obtuse angle allows us to place the to-be stacked vertices $w_1, \ldots w_k$ in B_e on a circular arc around u such that, for any $1 \leq i, j \leq k$, there exists a strongly monotone path between w_i and w_j; see Figure 3(c). Due to our invariant, there also exists a strongly monotone path between any of the newly stacked vertices and any vertex of an extension of the previous drawing with bubbles. Hence, after removing the bubble B_e, the resulting drawing is a strongly monotone drawing with bubbles.

In order to maintain the invariant, it remains to describe how to define the bubbles for the new active edges incident to the stacked vertices. For this purpose, we state the following Lemma 5, which enables us to define the two bubbles for the edges incident to any degree-2 vertex with an obtuse angle. The Lemma is then iteratively applied to the vertices $w_1, \ldots, w_k$ and after every usage of the Lemma the produced drawing with bubbles is strongly monotone. This iterative approach is used to ensure that, when defining bubbles for some vertex w_i, the previously added bubbles for $w_1, \ldots, w_{i-1}$ are taken into account.

Lemma 5 *Let Γ be a strongly monotone drawing with bubbles and let w be a vertex of degree 2 with an obtuse angle such that the two incident edges $e_1 = (u, w)$ and $e_2 = (v, w)$ have no bubbles. Then, there exist bubbles B_{e_1} and B_{e_2} for edges e_1 and e_2 respectively that only allow obtuse angles such that Γ remains strongly monotone with bubbles if we add B_{e_1} and B_{e_2}.*

References

[1] P. Angelini, E. Colasante, G. D. Battista, F. Frati, and M. Patrignani. Monotone drawings of graphs. *J. Graph Algorithms Appl.*, 16(1):5–35, 2012.

[2] G. R. Brightwell and E. R. Scheinerman. Representations of planar graphs. *SIAM Journal on Discrete Mathematics*, 6(2):214–229, 1993.

[3] S. Felsner, A. Igamberdiev, P. Kindermann, B. Klemz, T. Mchedlidze, and M. Scheucher. Strongly monotone drawings of planar graphs. arXiv:1601.01598, 2016.

[4] M. I. Hossain and M. S. Rahman. Monotone grid drawings of planar graphs. In J. Chen, J. E. Hopcroft, and J. Wang, editors, *Proc. 8th Int. Workshop Front. Algorithmics (FAW'14)*, volume 8497 of *Lecture Notes Comput. Sci.*, pages 105–116. Springer, 2014.

[5] P. Kindermann, A. Schulz, J. Spoerhase, and A. Wolff. On monotone drawings of trees. In C. Duncan and A. Symvonis, editors, *Proc. 22nd Int. Symp. Graph Drawing (GD'14)*, volume 8871 of *Lecture Notes Comput. Sci.*, pages 488–500. Springer, 2014.

[6] M. Nöllenburg, R. Prutkin, and I. Rutter. On self-approaching and increasing-chord drawings of 3-connected planar graphs. arXiv:1409.0315, 2014.

1-bend Upward Planar Drawings of SP-digraphs with the Optimal Number of Slopes*

Emilio Di Giacomo† Giuseppe Liotta† Fabrizio Montecchiani†

Abstract

It is proved that every series-parallel digraph whose maximum vertex-degree is Δ admits an upward planar drawing with at most one bend per edge such that each segment along each edge has one of Δ distinct slopes. This is shown to be worst-case optimal in terms of the number of slopes. Furthermore, our construction gives rise to drawings with optimal angular resolution $\frac{\pi}{\Delta}$.

1 Introduction

The *k-bend planar slope number* of a family of planar graphs with maximum vertex-degree Δ is the minimum number of distinct slopes used for the edges when computing a crossing-free drawing with at most $k > 0$ bends per edge of any graph in the family. For example, if $\Delta = 4$, a classic result is that every planar graph has a crossing-free drawing such that every edge segment is either horizontal or vertical and each edge has at most two bends (see, e.g., [1]). Clearly this bound on the number of slopes is optimal. This result has been extended to values of Δ larger than four by Keszegh et al. [5], who prove that $\lceil \frac{\Delta}{2} \rceil$ slopes suffice to construct a planar drawing with at most two bends per edge for any planar graph.

However if additional geometric constraints are imposed on the crossing-free drawing, only a few tight bounds on the planar slope number are known. For example, if one requires that the edges cannot have bends, the best known upper bound on the planar slope number is $O(c^{\Delta})$ (for a constant $c > 1$) while a general lower bound of just $3\Delta - 6$ has been proved [5]. Tight bounds are only known for outerplanar graphs [6] and subcubic graphs [3], while the gap between upper and lower bound has been reduced for planar graphs with treewidth two [8] or three [4]. If one bend per edge is allowed, Keszegh et al. [5] show an upper bound of 2Δ and a lower bound of $\frac{3}{4}(\Delta - 1)$ on the planar slope number of the planar graphs with maximum vertex-degree Δ. In a recent paper, Knauer and Walczak [7] lower the upper bound to $\frac{3}{2}(\Delta - 1)$; in the same paper, it is also proved that a tight bound of $\lceil \frac{\Delta}{2} \rceil$ can be achieved for outerplanar graphs.

In this paper we focus on the 1-bend planar slope number of directed graphs with the additional requirement that the computed drawings are *upward*, i.e., each edge is drawn as a curve monotonically increasing in the y-direction. We recall that upward drawings are a classic research topic in graph drawing, see e.g. [2]. We show that every series-parallel digraph G whose maximum vertex-degree is Δ has *1-bend upward planar slope number* Δ. That is, G admits an upward planar drawing with at most one bend per edge where at most Δ distinct slopes are used for the edges. This is shown to be worst-case optimal in terms of the number of slopes.

To prove the above results, we first construct a suitable contact representation of a series-parallel (di)graph where each vertex is represented as a cross, i.e. a horizontal segment properly intersected by a vertical segment (Section 3); then, we transform such contact representation into a 1-bend (upward) planar drawing optimizing the number of slopes used in such transformation (Section 4). Our construction gives rise to drawings with optimal angular resolution (i.e. the minimum angle between any two consecutive edges around a vertex); namely, the angular resolution is at least $\frac{\pi}{\Delta}$. Preliminaries are in Section 2. We conclude with some open problems in Section 5.

2 Preliminaries

A *series-parallel digraph* (*SP-digraph* for short) [2] is a simple planar digraph that has one source and one sink, called *poles*, and it is recursively defined as follows. A single edge is an SP-digraph. The digraph obtained by identifying the sources and the sinks of two SP-digraphs is an SP-digraph (*parallel composition*). The digraph obtained by identifying the sink of one SP-digraph with the source of a second SP-digraph is an SP-digraph (*series composition*). A *reduced* SP-digraph is a SP-digraph without transitive edges. The underlying undirected graph of an SP-digraph is called an *SP-graph*. An SP-digraph G is naturally associated with a binary tree T, which is called the *decomposition tree* of G. The nodes of T are of three types, *Q-nodes*, *S-nodes*, and *P-nodes*, rep-

*Research supported in part by the MIUR project AMANDA "Algorithmics for MAssive and Networked DAta", prot. 2012C4E3KT_001.

†Dipartimento di Ingegneria, Università degli Studi di Perugia, Italy, {emilio.digiacomo, giuseppe.liotta, fabrizio.montecchiani}@unipg.it

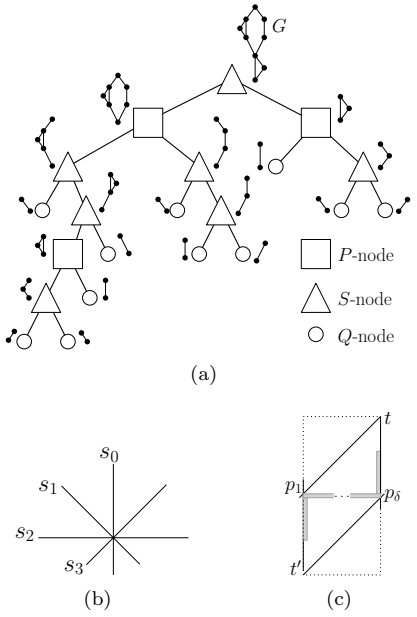


Figure 1: (a) An SP-graph G and its decomposition tree. (b) The slope-set $\mathcal{S}_4$. (c) The safe-region (dotted) of a cross (when Δ=4).

resenting single edges, series compositions, and parallel compositions, respectively. An SP-graph G and its decomposition tree are shown in Fig. 1(a). The decomposition tree of G has $O(n)$ nodes and can be constructed in $O(n)$ time [2].

The *slope* s of a line ℓ is the angle that a horizontal line needs to be rotated counter-clockwise in order to make it overlap with ℓ. The slope of a segment is the slope of the supporting line containing it. We denote by $\mathcal{S}_\Delta$ the set of slopes: $s_i = \frac{\pi}{2} + i\frac{\pi}{\Delta}$ $(i = 0, \ldots, \Delta - 1)$ (see Fig. 1(b)).

3 Cross Contact Representations

Basic definitions. A *cross* is composed of one horizontal segment and one vertical segment that share an interior point, the *center* of the cross. A cross is *degenerate* if either its horizontal or its vertical segment has zero length. The center of a degenerate cross is its midpoint. A point p of a cross c is an *end-point* (*interior point*) of c if it is an end-point (interior point) of the horizontal or vertical segment of c. Two crosses c_1 and c_2 *touch* if they share a point p, called *contact*, such that p is an end-point of the vertical (horizontal) segment of c_1 and an interior point of the horizontal (vertical) segment of c_2. A *cross-contact representation (CCR)* of a graph G, is a drawing γ such that: (i) Every vertex v of G is represented by a (possibly degenerate) cross $c(v)$; (ii) All intersections of crosses are touches, and (iii) Two crosses $c(u)$ and $c(v)$ touch if and only if G contains the edge (u, v).

We now consider CCRs of digraphs, and define properties that will be useful to transform the computed CCR into a 1-bend upward drawing of the corresponding digraph with few slopes and good angular resolution. Let γ be a CCR of a digraph G with maximum vertex-degree Δ. Let (u, v) be an edge of G oriented from u to v. Let p be the contact between $c(u)$ and $c(v)$. The point p is an *upward contact* if the following two conditions hold: (a) p is an end-point of the vertical segment of one of the two crosses and an interior point of the other cross, and (b) the center of $c(v)$ is above the center of $c(u)$. A CCR of a digraph G such that all its contacts are upward is an *upward CCR (UCCR)*. An UCCR γ is *balanced* if for every non-degenerate cross $c(u)$ of γ, we have that $|n_l(u) - n_r(u)| \leq 1$, where $n_l(u)$ $(n_r(u))$ is the number of contacts on the left (right) of the center of $c(u)$. Also, let $\{p_1, p_2, \ldots, p_\delta\}$ be the $\delta \geq 1$ contacts along the horizontal segment of $c(u)$, in this order from the leftmost one (p_1) to the rightmost one (p_δ). Let t be the intersection point between the vertical line passing through p_δ and the line with slope $\frac{\pi}{2} - \frac{\pi}{\Delta}$ and passing through p_1. Similarly, let t' be the intersection point between the vertical line passing through p_1 and the line with slope $\frac{\pi}{2} - \frac{\pi}{\Delta}$ and passing through p_δ. The *safe-region* of $c(u)$ is the rectangle having t and t' as the top-right and bottom-left corner, respectively. See Fig. 1(c) for an illustration. If $\delta = 1$, the safe-region degenerates to a point. An UCCR γ is *well-spaced* if no two safe-regions intersect each other.

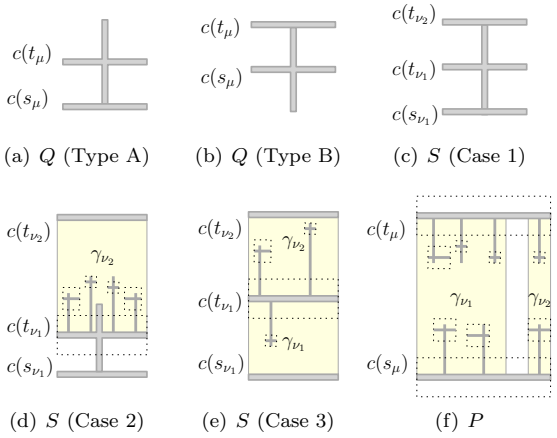
Algorithm overview. In the remainder of this section we describe a linear-time algorithm, `UCCRDrawer`, that takes as input a reduced SP-digraph G, and computes an UCCR γ of G which is balanced and well-spaced. The algorithm computes γ through a bottom-up visit of the decomposition tree T of G. For each node μ of T, `UCCRDrawer` computes an UCCR γ_μ of the graph G_μ associated with μ such that the following properties hold:

P1. γ_μ is balanced.

P2. γ_μ is well-spaced.

P3. Let s_μ and t_μ be the two poles of G_μ. If μ is a P- or an S-node, then γ_μ is contained in a rectangle R_μ such that its bottomost (topmost) side is the cross representing $c(s_\mu)$ $(c(t_\mu))$, which is therefore degenerate.

Drawing construction. As already said, `UCCRDrawer` computes γ through a bottom-up visit of the decomposition tree T of G. For each leaf node μ (which is a Q-node) the associated graph G_μ consists of a single edge (s_μ, t_μ). In this case, we define two possible types of UCCR, γ_μ^A (type A) and γ_μ^B (type B), of G_μ, as in Figs. 2(a) and 2(b). Properties **P1** – **P2** trivially hold in this case, while property **P3** does not apply.



(a) Q (Type A) (b) Q (Type B) (c) S (Case 1)

(d) S (Case 2) (e) S (Case 3) (f) P

Figure 2: Illustration for algorithm `UCCRDrawer`. The safe-regions are dotted (they are not in scale).

For each non-leaf node μ of T, `UCCRDrawer` computes the UCCR γ_μ by suitably combining the (already) computed UCCRs γ_{ν_1} and γ_{ν_2} of the two graphs associated with the children ν_1 and ν_2 of μ. If μ is an S-node of T, we distinguish between the following cases, where we assume that $t_{\nu_1} = s_{\nu_2}$ is the pole shared by ν_1 and ν_2.

Case 1. Both ν_1 and ν_2 are Q-nodes. Then an UCCR of G_μ is computed by combining $\gamma^A_{\nu_1}$ and $\gamma^B_{\nu_2}$ as in Fig. 2(c). Properties **P1** − **P3** trivially hold.

Case 2. ν_1 is a Q-node , while ν_2 is not (the case when ν_2 is a Q-node and ν_1 is not, is symmetric). We combine the drawing $\gamma^A_{\nu_1}$ of G_{ν_1} and the drawing γ_{ν_2} of G_{ν_2} as in Fig. 2(d). Notice that, to combine the two drawings we may need to scale one of them so that their widths are the same. To ensure **P1**, we move the vertical segment s of $c(t_{\nu_1}) = c(s_{\nu_2})$ so that $|n_l(t_{\nu_1}) - n_r(t_{\nu_1})| \leq 1$. We may also need to shorten its upper part in order to avoid crossings with other segments, and to extend its lower part so that $c(s_{\nu_1})$ is outside the safe-region of $c(t_{\nu_1}) = c(s_{\nu_2})$, thus guaranteeing property **P2**. Property **P3** holds by construction.

Case 3. If none of ν_1 and ν_2 is a Q-node, then we combine γ_{ν_1} and γ_{ν_2} as in Fig. 2(e). Also in this case we may need to scale one of the two drawings so that their widths are the same. Property **P1** holds, as it holds for γ_{ν_1} and γ_{ν_2}. Property **P2** may not hold for $c(t_{\nu_1}) = c(s_{\nu_2})$. We can ensure **P2** by performing the following stretching operation. Let ℓ_a and ℓ_b be two horizontal lines slightly above and slightly below the horizontal segment of $c(t_{\nu_1}) = c(s_{\nu_2})$, respectively. We extend all the vertical segments intersected by ℓ_a or ℓ_b until the safe-region of $c(t_{\nu_1}) = c(s_{\nu_2})$ does not intersect any other safe-region. Property **P3** holds by construction.

Finally, suppose that μ is a P-node of T, having ν_1 and ν_2 as children (recall that neither ν_1 nor ν_2 is a Q-node, since G is a reduced SP-digraph). We combine γ_{ν_1} and γ_{ν_2} as in Fig. 2(f). We may need to scale one of the two drawings so that their heights are the same. Property **P1** holds, as it holds for γ_{ν_1} and γ_{ν_2}. To ensure **P2**, a stretching operation similar to the one described in Case 3 is possibly performed by using a horizontal line slightly above (below) the horizontal segment of $c(s_\mu)$ ($c(t_\mu)$). Property **P3** holds by construction.

It is possible to show that algorithm `UCCRDrawer` can be implemented to run in linear time. The above discussion can be used to prove the following.

Lemma 1 *Let G be an n-vertex reduced SP-digraph. Algorithm `UCCRDrawer` computes a balanced and well-spaced UCCR γ of G in $O(n)$ time.*

4 1-bend Upward Planar Drawings

In this section we first describe how to transform an UCCR of a reduced SP-digraph into an upward 1-bend planar drawing that uses the slopes in the slope-set $\mathcal{S}_\Delta$. We then explain how to deal with general SP-digraphs.

Let γ be an UCCR of a reduced SP-digraph G and let $c(u)$ be the cross representing a vertex u of G in γ. Let $p_1, \ldots, p_\delta$ ($\delta \geq 1$) be the contacts along the horizontal segment of $c(u)$. We assume that these contacts are ordered such that we first have all the contacts corresponding to the outgoing edges of u from left to right, and then we have all the contacts corresponding to the incoming edges of u from right to left. Let c be either the center of $c(u)$, if $c(u)$ is non-degenerate, or $p_{\lfloor \frac{\delta}{2} \rfloor + 1}$ if $c(u)$ is degenerate. Consider the set of lines $\ell_0, \ldots, \ell_{\Delta-1}$, such that ℓ_i passes through c and has slope $s_i \in \mathcal{S}_\Delta$ (for $i = 0, \ldots, \Delta - 1$). These lines intersect all the vertical segments forming a contact on the horizontal segment of $c(u)$. In particular, each quadrant of $c(u)$ contains a number of lines that is at least the number of vertical segments touching $c(u)$ in that quadrant. Since γ is well-spaced, all these intersections are inside the safe-region of $c(u)$. Hence we can replace each contact of $c(u)$ with two segments having slope in $\mathcal{S}_\Delta$ as shown in Fig. 3(a) and 3(b). More precisely, each contact p_i of $c(u)$ is replaced with two segments that are both in the quadrant of $c(u)$ that contains the vertical segment defining p_i. This guarantees the upwardness of the resulting drawing. Also, each edge has at most one bend. Namely, each edge is represented by a single contact between a horizontal and a vertical segment and we introduce one bend only when dealing with the cross containing the horizontal segment. Finally, Γ is planar. Namely, there is no crossing in γ and each cross is only modified locally inside its safe-region which, by the well-spaced property, is disjoint by any other safe-region.

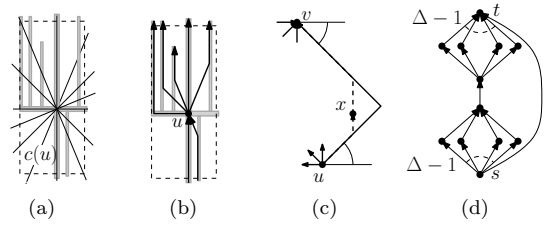


Figure 3: (a)-(b) Transforming an UCCR into a 1-bend drawing. (c) Drawing of a transitive edge. (d) A SP-digraph requiring at least Δ slopes in any 1-bend upward planar drawing.

Using the technique described above, we can compute an 1-bend upward planar drawing with slopes in the slope-set $\mathcal{S}_\Delta$ for reduced SP-digraphs. We now explain how to deal with a general SP-digraph G. First, we change the embedding of G as follows. Let (u, v) be a transitive edge, and let G' be the maximal subgraph of G having u and v as poles. We change the embedding of G' so that (u, v) is the rightmost outgoing edge of u and the rightmost incoming edge of v. Second, we subdivide (u, v) with a dummy vertex x. The resulting graph G_r is a reduced SP-digraph and therefore we can compute an 1-bend upward planar drawing Γ_r of G_r as described above. When doing so, we take care of guaranteeing that the drawings of (u, x) and (x, v) (for each transitive edge (u, v)) do not use the horizontal slope (it is not hard to see that this is always possible). Each transitive edge (u, v) of G is represented in Γ_r by a path of two edges (u, x) and (x, v). If at least one between (u, x) and (x, v) is drawn with no bend, then it is sufficient to remove x to obtain a 1-bend drawing of (u, v). If both (u, x) and (x, v) have one bend, then simply removing the subdivision vertex would lead to a 2-bend drawing of (u, v). In this case we have to modify the drawing of (u, v). Let ℓ_u be the straight line passing through u and the bend of (u, x) and let ℓ_v be the straight line passing through v and the bend of (x, v). We obtain a 1-bend drawing of (u, v) by placing a single bend at the intersection point of ℓ_u and ℓ_v (see Figure 3(c)). Since we did not use the horizontal slope in the drawing of (u, x) and (x, v) such a point exists. With this operation, the drawing of (u, v) has been extended to the right, and it is possible to modify the construction of the UCCR γ so that (u, v) does not cross any other edge. The modification of `UCCRDrawer` is such that when a P-node is processed, it additionally ensures the existence of an empty region where (u, v) can be drawn without crossings. Details are omitted.

We conclude by exhibiting in Fig. 3(d) a family of SP-digraphs, such that, for every value of Δ, there exists a graph in this family with maximum vertex-degree Δ and that requires at least Δ slopes in any 1-bend upward planar drawing. Namely, if a graph G has a source (or a sink) of degree Δ, then it requires at least $\Delta - 1$ slopes in any upward drawing because each slope, with the only possible exception of the horizontal one, can be used for a single edge. In the graph of Fig. 3(d) however, the edge (s, t) must be either the leftmost or the rightmost edge of s and t in any upward drawing. Therefore, if only $\Delta - 1$ slopes are allowed, such edge cannot be drawn planarly and with one bend. Thus, the following theorem holds.

Theorem 2 *Every n-vertex SP-digraph G with maximum vertex-degree Δ admits a 1-bend upward planar drawing Γ with at most Δ slopes and angular resolution at least $\frac{\pi}{\Delta}$. These bounds on the number of slopes and on the angular resolution are worst-case optimal. Also, Γ can be computed in $O(n)$ time.*

Since every SP-graph can be oriented to an SP-digraph, next corollary is implied by Theorem 2 and lowers the upper bound for planar graphs in [7].

Corollary 1 *The 1-bend planar slope number of SP-graphs with maximum vertex-degree Δ is at most Δ.*

5 Conclusions and Open Problems

We proved that the 1-bend upward planar slope number of SP-digraphs with maximum vertex-degree Δ is at most Δ and this is a tight bound. Is the bound of Corollary 1 also tight? Moreover, can it be extended to any partial 2-tree?

References

[1] T. C. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *CGTA*, 9(3):159–180, 1998.

[2] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice-Hall, 1999.

[3] E. Di Giacomo, G. Liotta, and F. Montecchiani. The planar slope number of subcubic graphs. In *LATIN 2014*, volume 8392 of *LNCS*, pages 132–143. Springer, 2014.

[4] V. Jelínek, E. Jelínková, J. Kratochvíl, B. Lidický, M. Tesar, and T. Vyskocil. The planar slope number of planar partial 3-trees of bounded degree. *Graphs and Combinatorics*, 29(4):981–1005, 2013.

[5] B. Keszegh, J. Pach, and D. Pálvölgyi. Drawing planar graphs of bounded degree with few slopes. *SIAM J. Discrete Math.*, 27(2):1171–1183, 2013.

[6] K. B. Knauer, P. Micek, and B. Walczak. Outerplanar graph drawings with few slopes. *CGTA*, 47(5):614–624, 2014.

[7] K. B. Knauer and B. Walczak. Graph drawings with one bend and few slopes. *CoRR*, abs/1506.04423, 2015.

[8] W. Lenhart, G. Liotta, D. Mondal, and R. I. Nishat. Planar and plane slope number of partial 2-trees. In *GD 2013*, volume 8242 of *LNCS*, pages 412–423. Springer, 2013.

Improved Bounds on the Growth Constant of Polyiamonds*

Gill Barequet† Mira Shalah*

Abstract

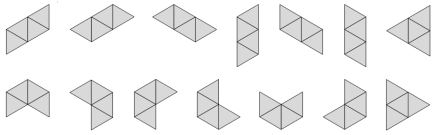
A polyiamond is an edge-connected set of cells on the triangular lattice. In this paper we provide improved lower and upper bounds on the asymptotic growth constant of polyiamonds, proving that it is between 2.8424 and 3.6050.

1 Introduction

(a) $n = 1$: Two moniamonds

(b) $n = 2$: Three diamonds

(c) $n = 3$: 6 triamonds



(d) $n = 4$: 14 tetriamonds

A *polyomino* of size n is an edge-connected set of n cells on the square lattice $\mathbb{Z}^2$. Similarly, a *polyiamond* of size n is an edge-connected set of n cells on the triangular lattice. *Fixed* polyiamonds are considered distinct if they have different *shapes* or *orientations*. In this paper we consider only fixed polyiamonds, and so we refer to them simply as "polyiamonds." Figure 1 shows polyiamonds of size 1–4.

In general, a connected set of cells on a lattice is called a *lattice animal*. The fundamental combinatorial problem concerning lattice animals is "How many animals with n cells are there?" The study of lattice animals began in parallel more than half a century ago in two different communities. In statistical physics, Temperley [19] investigated the mechanics of macromolecules, and Broadbent and Hammersley [5] studied percolation processes. In mathematics, Eden [6] and others analyzed cell growth problems. Since then, counting animals has attracted much attention in the literature. However, despite serious efforts over the last 50 years, counting polyominoes is still far from being solved, and is considered one of the long-standing open problems in combinatorial geometry.

The symbol $A(n)$ usually denotes the number of polyominoes of size n; See sequence A001168 in the On-line Encyclopedia of Integer Sequences (OEIS) [1]. Since no analytic formula for the number of animals is yet known for any nontrivial lattice, a great portion of the research has so far focused on efficient algorithms for *counting* animals on lattices, primarily on the square lattice. Elements of the sequence $A(n)$ are currently known up to $n = 56$ [11]. The growth constant of polyominoes was also treated extensively in the literature, and a few asymptotic results are known. Klarner [12] showed that the limit $\lambda := \lim_{n\to\infty} \sqrt[n]{A(n)}$ exists, and the main problem so far has been to evaluate this constant. The convergence of $A(n+1)/A(n)$ to λ (as $n\to\infty$) was proven only three decades later by Madras [15], using a novel pattern-frequency argument. The best-known lower and upper bounds on λ are 4.0025 [4] and 4.6496 [13], respectively. It is widely believed (see, e.g., [7, 8]) that $\lambda \approx 4.06$, and the currently best estimate, $\lambda = 4.0625696 \pm 0.0000005$, is due to Jensen [11].

In the same manner, let $T(n)$ denote the number of polyiamonds of size n (sequence A001420 in the OEIS). Elements of the sequence $T(n)$ were computed up to $n = 75$ [9, p. 479] using a transfer-matrix algorithn by Jensen [ibid., p. 173], adapting his original polyomino-counting algorithm [11]. Earlier counts were given by Lunnon [14] up to size 16, by Sykes and Glen [18] up to size 22, and by Aleksandrowicz and Barequet [2] (extending Redelmeier's polyomino-counting algorithm [17]) up to size 31.

Similarly to polyominoes, the limits $\lim_{n\to\infty} \sqrt[n]{T(n)}$ and $\lim_{n\to\infty} T(n+1)/T(n)$ exist and are equal. Let, then, λ_T denote the growth constant of polyiamonds. Klarner [12, p. 857] showed that $\lambda_T \geq 2.13$ by taking the square root of 4.54, a lower bound he computed for the growth constant of animals on the rhomboidal lattice, using the fact that a rhombus is made of two neighboring equilateral triangles. This bound is also mentioned by Lunnon [14, p. 98]. Rands and Welsh [16] used

*Work on this paper by both authors has been supported in part by ISF Grant 575/15.

†Dept. of Computer Science, The Technion, Haifa 32000, Israel. E-mail: {barequet,mshalah}@cs.technion.ac.il

renewal sequences in order to show that

$$\lambda_T \geq (T(n)/(2(1+\lambda_T)))^{1/n} \quad (1)$$

for any $n \in \mathbb{N}$. Substituting the easy upper bound $\lambda_T \leq 4$ (see below) in the right-hand side of this relation, and knowing at that time elements of the sequence $T(n)$ for $1 \leq n \leq 20$ only (data provided by Sykes and Glen [18]), they used $T(20) = 173,338,962$ to show that $\lambda_T \geq (T(20)/10)^{1/20} \approx 2.3011$. Nowadays, since we know $T(n)$ up to $n = 75$,[1] we can obtain, using the same method, that $\lambda_T \geq (T(75)/10)^{1/75} \approx 2.7714$. We can even do slightly better than that. Substituting in Equation (1) the upper bound we obtain in Section 3 ($\lambda_T \leq 3.6050$), we see that $\lambda_T \geq (T(75)/(2(1+3.6050)))^{1/75} \approx 2.7744$. However, we can still improve on this.

An easy upper bound, based on an idea of Eden [6] (originally applied to the square lattice for setting an upper bound on λ), was described by Lunnon [14, p. 98]. Every polyiamond P can be built according to a set of $n-1$ "instructions" taken from a superset of size $2(n-1)$. Each instruction tells us how to choose a lattice cell c, neighboring a cell already in P, and add c to P. (Some of these instruction sets are illegal, and other sets produce the same polyiamonds, but this only helps.) Hence, $\lambda_T \leq \lim_{n\to\infty} \binom{2(n-1)}{n-1}^{1/n} = 4$.

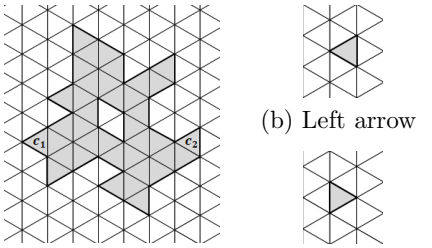
As can be seen, there is a large gap between the lower and upper bounds on λ_T. Based on existing data, it is believed [18] (but has never been proven) that $\lambda_T = 3.04 \pm 0.02$. In this paper we improve both lower and upper bounds on λ_T, showing that $2.8424 \leq \lambda_T \leq 3.6050$. The new lower bound is obtained by using a concatenation argument tailored to the triangular lattice, and the new upper bound is obtained by investigating the growth constant of a sequence dominating the enumerating sequence of polyiamonds.

2 Lower Bound

A concatenation of two polyiamonds P_1, P_2 is the translation of P_1 relative to P_2, so that P_1, P_2 do not overlap but their union is a valid (connected) polyiamond, and all the translated versions of the cells of P_1 are smaller than the cells of P_2 under a proper definition of a lexicographic order on the cells of the lattice. We use a concatenation argument in order to improve the lower bound on λ_T.

Theorem 1 $\lambda_T \geq 2.8424$.

Proof. We orient the triangular lattice as is shown in Figure 1(a), and define a lexicographic order on the cells of the lattice as follows: A cell c_1 is *smaller* than cell $c_2 \neq c_1$ (denoted as $c_1 < c_2$) if the lattice column of c_1 is to the left of the column of c_2, or if c_1, c_2 are in the same column and c_1 is below c_2. Denote triangles which look like a "left arrow" (Figure 1(b)) as triangles of Type 1, and triangles which look like a "right arrow" (Figure 1(c)) as triangles of Type 2. Let $T_1(n)$ be the number of polyiamonds of size n whose *largest* (top-right) triangle is of Type 1, and let $T_2(n)$ be the number of polyiamonds of size n whose *largest* triangle is of Type 2. Obviously, we have $T(n) = T_1(n) + T_2(n)$.[2] An interesting observation is that by rotational symmetry, the number of polyiamonds of size n, whose *smallest* (bottom-left) triangle is of Type 2, is also $T_1(n)$, and so the number of polyiamonds, whose *smallest* triangle is of Type 1, is $T_2(n)$.



(a) A polyiamond (b) Left arrow (c) Right arrow

Figure 1: Polyiamonds on the triangular lattice

We now proceed with a standard concatenation argument, tailored to the specific case of the triangular lattice. Interestingly, not all pairs of polyiamonds of size n can be concatenated. In addition, there exist many polyiamonds of size $2n$ which cannot be represented as the concatenation of two polyiamonds of size n. Let us count carefully the amount of pairs of polyiamonds that can be concatenated.

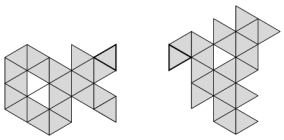
- Polyiamonds, whose largest triangle is of Type 1, can be concatenated only to polyiamonds whose smallest triangle is of Type 2, and this can be done in two different ways (see Figure 2(a)). There are $2(T_1(n))^2$ concatenations of this kind.
- Polyiamonds, whose largest triangle is of Type 2, can be concatenated, in a single way, only to polyiamonds whose smallest triangle is of Type 1 (see Figure 2(b)). There are $(T_2(n))^2$ concatenations of this kind.

Altogether, we have $2(T_1(n))^2+(T_2(n))^2$ possible concatenations, and, as argued above,

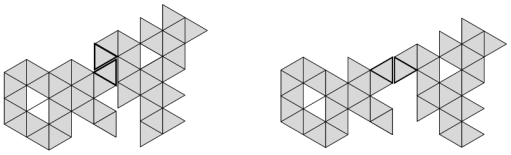
$$2(T_1(n))^2 + (T_2(n))^2 \leq T(2n). \quad (2)$$

[1] $T(75) = 15,936,363,137,225,733,301,433,441,827,683,823$.

[2] Observe that $T_1(n) = T_2(n-1)$ and, hence, $T(n) = T_2(n)+T_2(n-1)$. Indeed, when the largest cell of a polyiamond P is of Type 1, its only possible neighboring cell within P is the cell immediately below it. Therefore, the number of polyiamonds of size n whose largest cell is of Type 1 is equal to the number of polyiamonds of size $n-1$ whose largest cell is of Type 2.

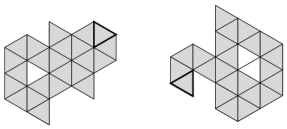


Two polyiamonds

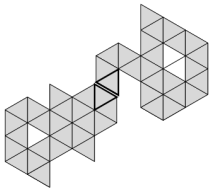


Vertical concatenation Horizontal concatenation

(a) Two concatenations of Type-1 and Type-2 triangles



Two polyiamonds



Vertical concatenation

(b) A single concatenation of Type-2 and Type-1 triangles

Figure 2: Possible concatenations of polyiamonds

Let us now find a lower bound on the number of concatenations. Let $x = x(n)$ be the fraction of polyiamonds of Type 1 out of all polyiamonds of size n, i.e., $T_1(n) = xT(n)$ and $T_2(n) = (1-x)T(n)$. Eq. (2) can then be rewritten as $T(2n) \geq 2(xT(n))^2 + ((1-x)T(n))^2 = (3x^2 - 2x + 1)T^2(n)$. Elementary calculus shows that the function $f(x) = 3x^2 - 2x + 1$ assumes its minimum at $x = 1/3$ and that $f(1/3) = 2/3$. Hence,

$$\frac{2}{3}T^2(n) \leq T(2n).$$

By manipulating this relation, we obtain that

$$\left(\frac{2}{3}T(n)\right)^{1/n} \leq \left(\frac{2}{3}T(2n)\right)^{1/(2n)}$$

This implies that the sequence $\left(\frac{2}{3}T(k)\right)^{1/k}$, $\left(\frac{2}{3}T(2k)\right)^{1/(2k)}$, $\left(\frac{2}{3}T(4k)\right)^{1/(4k)}, \ldots$ is monotone increasing for any value of k, and, as a subsequence of $\left(\left(\frac{2}{3}T(n)\right)^{1/n}\right)$, it converges to λ_T too. Therefore, any term of the form $\left(\frac{2}{3}T(n)\right)^{1/n}$ is a lower bound on λ_T. In particular, $\lambda_T \geq (\frac{2}{3}T(75))^{1/75} \approx 2.8424$. □

3 Upper Bound

We follow the method used recently [3] for polyominoes (animals on the *square* lattice).[3]

3.1 Number of Compositions

Definition 2 *A polyiamond P can be* decomposed *into two polyiamonds P_1, P_2 if the cell set of P can be split into two complementing non-empty subsets, such that each subset is a valid (connected) polyiamond. We also say that the polyiamonds P_1, P_2 can be* composed *so as to yield the polyiamond P.*

A *composition* of two polyiamonds is a natural generalization of the widely-used notion of the *concatenation* of polyiamonds. In fact, concatenation is a composition in lexicographic order.

Theorem 3 *(Composition) Let P_1, P_2 be two polyiamonds of sizes n_1 and n_2, respectively. Then, P_1 and P_2 can be composed and yield at most $(n_1+2)(n_2+2)/2$ different polyiamonds.*[4]

Proof. Refer again to Figure 1(a). A boundary edge of a polyiamond can be either vertical, ascending, or descending. The inside of the polyiamond can be either to the left or to the right of a boundary edge,[5] where the latter case is marked below by overlining. Denote, then, the number of boundary edges of the various types by x and $\bar{x}$, where $x \in \{v, a, d\}$. Thus, if the perimeter of a polyiamond is p, we can classify its boundary by the vector $(v, a, d, \bar{v}, \bar{a}, \bar{d})$, where $v+a+d+\bar{v}+\bar{a}+\bar{d} = p$. Suppose we are given two polyiamonds P_1, P_2 with respective perimeters p_1, p_2 and associated perimeter vectors $(v_i, a_i, d_i, \bar{v_i}, \bar{a_i}, \bar{d_i})$ (for $i = 1, 2$). Then, a trivial upper bound on the number of compositions of P_1, P_2 is $\sum_{t \in \{v,a,d,\bar{v},\bar{a},\bar{d}\}} t_1 \bar{t_2}$, using the convention $\bar{\bar{t_i}} = t_i$. Note that the number of boundary edges of any type of a polyiamond of perimeter p cannot exceed $p/2$. Therefore, by convexity, the number of compositions of P_1, P_2 is bounded from above by $2(p_1/2 \cdot p_2/2) = p_1 p_2/2$. (A slightly sharper upper bound which takes into account odd perimeters is $\lceil p_1/2 \rceil \lceil p_2/2 \rceil + \lfloor p_1/2 \rfloor \lfloor p_2/2 \rfloor$.) The perimeter of a polyiamond of size n is maximized when the cell-adjacency graph of the polyiamond is a tree, in which case the perimeter is $n+2$. (Indeed, the perimeter of a single triangle is 3, and each of the additional $n-1$ triangles adds at most 1 to the perimeter.) The claim follows. □

[3] There is a gap in Theorem 3 in this reference. Therefore, we take a different approach in Theorem 3 below.

[4] A slightly sharper upper bound is $(\lceil n_1/2 \rceil + 1)(\lceil n_2/2 \rceil + 1) + (\lfloor n_1/2 \rfloor + 1)(\lfloor n_2/2 \rfloor + 1)$.

[5] The "left" (resp., "right") side of an ascending edge means *above* (resp., *below*) the edge, while the "left" (resp., "right") side of a descending edge means *below* (resp., *above*) the edge.

3.2 Balanced Decompositions

Definition 4 *A decomposition of a polyiamond of size n into two polyiamonds P_1, P_2 is* k-balanced *if $k \le |P_i| \le n-k$ (for $i = 1, 2$).*

Theorem 5 *Every polyiamond of size n has at least one $\lceil (n-1)/3 \rceil$-balanced decomposition.*

Proof. Let us rephrase the claim in graph terminology. In fact, we prove a stronger claim which states that every connected graph G, for which $\Delta(G) \le 3$, can be partitioned into two vertex-disjoint subgraphs G_1, G_2, such that (1) G_1, G_2 are connected; and (2) $\lceil (n-1)/3 \rceil \le |G_i| \le \lfloor (2n+1)/3 \rfloor$ (for $i = 1, 2$). This can be done constructively by considering a spanning tree of G, marking an arbitrary vertex as its root, and traversing the tree downwards from the root while keeping the invariant that either the already-traversed subgraph meets the size requirement or the untraversed part contains a subgraph with this property. When the process stops, which must be the case, the desired decomposition is found. □

3.3 The Bound

Theorem 6 $\lambda_T \le 3.6050$.

Proof. First, the combination of Theorems 3 and 5 implies that

$$T(n) \le \sum_{k=\lceil \frac{n-1}{3} \rceil}^{\lfloor n/2 \rfloor} (1 - \frac{\delta_{k,n/2}}{2}) \frac{(k+2)(n-k+2)}{2} T(k) T(n-k).$$

Indeed, every polyiamond of size n can be decomposed in at least one $\lceil (n-1)/3 \rceil$-balanced way into a pair of polyiamonds P_1, P_2 of sizes n_1 and n_2, respectively (where $n_1 + n_2 = n$), and a code with up to $(n_1+2)(n_2+2)/2$ options will tell us uniquely how to compose P_1, P_2 in order to reconstruct P. (The factor $(1 - \delta_{k,n/2}/2)$ compensates for double counting which occurs when P_1, P_2 are of the same size.) Naturally, P can be decomposed in more than one way, and the number of compositions of P_1, P_2 can be smaller than $(n_1+2)(n_2+2)/2$, but this only helps.

Second, define the sequence $T'(n)$ as follows.

$$T'(n) = \begin{cases} T(n) & 1 \le n \le 75; \\ \sum_{k=\lceil \frac{n-1}{3} \rceil}^{\lfloor n/2 \rfloor} (1 - \frac{\delta_{k,n/2}}{2}) \frac{(k+2)(n-k+2)}{2} T'(k) T'(n-k) & n > 75. \end{cases}$$

(Recall that the sequence $T(n)$ is known for $1 \le n \le 75$.) Since $T'(n) \ge T(n)$ for any value of $n \in \mathbb{N}$ (this can be proven by a simple induction on n), the growth constant of $T'(n)$, if it exists, is an upper bound on λ_T.

Numerical calculations show that $T'(n)$ does have an asymptotic growth constant which is about 3.6050, implying the claim. □

References

[1] *The On-Line Encyclopedia of Integer Sequences*, available at `http://oeis.org` .

[2] G. Aleksandrowicz and G. Barequet, Counting d-dimensional polycubes and nonrectangular planar polyominoes, *IJCGA*, **19**, 215–229, 2009.

[3] G. Barequet and R. Barequet, An improved upper bound on the growth constant of polyominoes, *Proc. EuroCOMB*, Bergen, Norway, Aug.-Sep. 2015, *ENDM*, 49, 167–172, November 2015.

[4] G. Barequet, G. Rote, and M. Shalah, $\lambda > 4$, *Proc. 23rd Ann. European Symp. on Algorithms*, Patras, Greece, *LNCS*, 9294, Springer-Verlag, 83–94, September 2015, *Comm. of the ACM*, to appear.

[5] S.R. Broadbent and J.M. Hammersley, Percolation processes: I. Crystals and mazes, *Proc. Cambridge Phil. Soc.*, **53**, 629–641, 1957.

[6] M. Eden, A two-dimensional growth process, *Proc. 4th Berkeley Symp. on Math. Statistics and Probability*, IV, Berkeley, CA, 223–239, 1961.

[7] D.S. Gaunt, The critical dimension for lattice animals, *J. of Physics, A: Math. and General*, **13**, L97–L101, 1980.

[8] D.S. Gaunt, M.F. Sykes, and H. Ruskin, Percolation processes in d-dimensions, *J. of Physics, A: Mathematical and General*, **9**, 1899–1911, 1976.

[9] A.J. Guttmann, *Polygons, Polyominoes, and Polycubes*, *Lecture Notes in Physics*, 775, Springer and Canopus Academic Publishing Ltd., 2009.

[10] F. Harary, Unsolved Problems in the enumeration of graphs, *Pub. of the Mathematical Inst. of the Hungarian Academy of Sciences*, **5**, 1–20, 1960.

[11] I. Jensen, Counting polyominoes: A parallel implementation for cluster computing, *Proc. Int. Conf. on Computational Science*, III (Melbourne, Australia and St. Petersburg, Russia, 2003), *LNCS*, 2659, Springer, 203–212.

[12] D.A. Klarner, Cell growth problems, *Canadian J. of Mathematics*, **19**, 851–863, 1967.

[13] D.A. Klarner and R.L. Rivest, A procedure for improving the upper bound for the number of n-ominoes, *Canadian J. of Mathematics*, **25**, 585–602, 1973.

[14] W.F. Lunnon, Couting hexagonal and triangular polyominoes, in: *Graph Theory and Computing* (R.C. Read, ed.), Academic Press, New York, 1972, 87–100.

[15] N. Madras, A pattern theorem for lattice clusters, *Annals of Combinatorics*, **3**, 357–384, 1999.

[16] B.M.I. Rands and D.J.A. Welsh, Animals, trees and renewal sequences, *IMA J. of Applied Mathemathics*, **27**, 1–17, 1981; Corrigendum, **28**, 107, 1982.

[17] D.H. Redelmeier, Counting polyominoes: Yet another attack, *Discrete Mathematics*, **36**, 191–203, 1981.

[18] M.F. Sykes and M. Glen, Percolation processes in two dimensions: I. Low-density series expansions, *J. of Physics, A: Mathematical and General*, **9**, 87–95, 1976.

[19] H.N.V. Temperley, Combinatorial problems suggested by the statistical mechanics of domains and of rubber-like molecules, *Physical Review 2*, **103**, 1–16, 1956.

Colouring Contact Graphs of Squares and Rectilinear Polygons

Mark de Berg* Aleksandar Markovic* Gerhard Woeginger*

Abstract

We study colourings of contact graphs of squares and rectilinear polygons. Our main results are that (i) it is NP-hard to decide if a contact graph of unit squares is 3-colourable, and (ii) any contact graph of a set of rectilinear polygons is 6-colourable.

1 Introduction

In graph-colouring problems the goal is to assign a colour to each node in a graph $\mathcal{G} = (V, E)$ such that the resulting colouring satisfies certain properties. The standard property is that for any edge $(u, v) \in E$ the nodes u and v have different colours. From now on, whenever we speak of a *colouring* of a graph we mean a colouring with this property. The minimum number of colours needed to colour a given graph is called the *chromatic number* of the graph. Two main questions regarding graph colouring are: (i) Given a graph $\mathcal{G}$ from a certain class of graphs, how quickly can we compute its chromatic number? (ii) What is the chromatic number of a given graph class, that is, the smallest number of colours such that any graph from the class can be coloured with that many colours?

We are interested in these questions for graphs induced by geometric objects in the plane and, in particular, by contact graphs. Let $\mathcal{S} = \{P_1, \ldots, P_n\}$ be a set of geometric objects in the plane. The *intersection graph* induced by $\mathcal{S}$ is the graph whose nodes correspond to the objects in $\mathcal{S}$ and where there is an edge (P_i, P_j) if and only if P_i and P_j intersect. If the objects in $\mathcal{S}$ are closed and have disjoint interiors, then the intersection graph is called a *contact graph*. It has been shown that the class of contact graphs of discs is the same as the class of planar graphs: any contact graph of discs is planar and any planar graph can be drawn as a contact graph of discs [6]. By the Four-Colour Theorem [1] this implies that any contact graph of discs is 4-colourable. More generally, contact graphs of compact objects with smooth boundaries are planar, and so they are 4-colourable.

We are interested in colouring contact graphs of squares and rectilinear polygons. (Unless explicitly stated otherwise, whenever we speak of squares or rectilinear polygons we mean axis-parallel squares and axis-parallel rectilinear polygons.) Contact graphs of squares are different from contact graphs of smooth objects, because four (interior-disjoint) squares can all meet in a common point. Thus the obvious embedding of such a contact graph—where we put a node at the center of each square and we connect the centers of two touching squares by a two-link path through a touching point—is not necessarily plane.

Eppstein *et al.* [4] studied colourings of contact graphs of squares for the special cases where the squares form a quadtree subdivision, that is, the set $\mathcal{S}$ of squares is obtained by recursively subdividing an initial square in four equal-sized quadrants. They proved that any such contact graph is 6-colourable and they gave an example of a quadtree subdivision that requires five colours. (They also considered the variant where two squares that only touch in a single vertex are not considered neighbours.)

Our results. We start by studying the computational complexity of colouring contact graphs. We show that already for a set of unit squares, it is NP-complete to decide if the contact graph is 3-colourable.

Next we study the chromatic number of various classes of contact graphs. Recall that the obvious embedding of the contact graph of squares need not be plane. We first prove contact graphs of unit squares can have a K_m as a minor for an arbitrarily large m and, hence, need not be planar. Nevertheless, contact graphs of unit squares are 4-colourable and finding a 4-colouring is quite easy, so our NP-completeness result on 3-colouring completely characterizes the computational complexity of colouring unit squares. Contact graphs of arbitrarily-sized squares are not always 4-colourable—the quadtree example of Eppstein *et al.* [4] requiring five colours shows this. We prove that the chromatic number of the class of contact graphs of arbitrarily-sized squares is at most 6. In fact, we prove that any contact graph of a set of rectilinear polygons is 6-colourable. (Even more generally, contact graphs of polygons whose interior angles are strictly greater than $2\pi/5$ are 6-colourable.) Thus we obtain the same bound of Eppstein *et al.*, but for a much larger class of objects. Moreover, for this class the bound is tight. To prove our result, we charaterize contact graphs of rectilinear polygons as a certain subset of 1-planar graphs, which are known to be 6-colourable [2].

*Department of Math and CS, TU Eindhoven. MdB, AM and GW are supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.

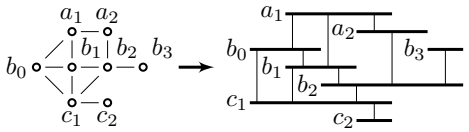
2 NP-Completeness of 3-Colourability

In this section we establish the hardness of 3-Colourability on contact graphs of unit squares.

Theorem 1 *3-Colourability on contact graphs of unit squares is NP-complete.*

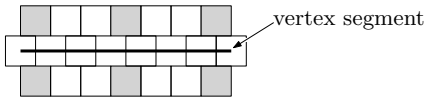
Proof. 3-Colourability on contact graphs is obviously in NP. To prove that the problem is NP-hard we use a reduction from the NP-complete problem of 3-colouring planar graphs of degree at most 4 [5].

Let $\mathcal{G} = (V, E)$ be any planar graph on n vertices with degree at most 4. Rosenstiehl and Tarjan [7] showed that we can compute in polynomial time a *visibility representation* of $\mathcal{G}$, in which every vertex $u \in V$ is represented by a horizontal *vertex segment* s_u and every edge $(u, v) \in E$ is represented by a vertical *edge segment* that connects s_u and s_v and does not intersect any other vertex segment.



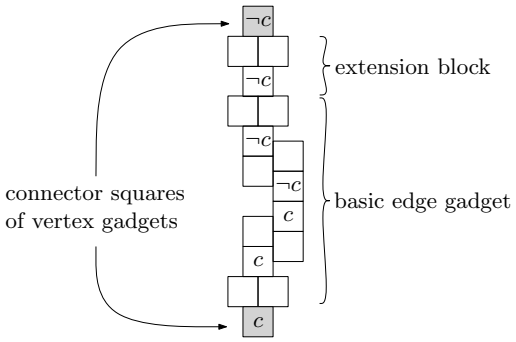
The construction can be done so that (i) all y-coordinates of the vertex segments are multiples of 10, and (ii) all x-coordinates of the edge segments are multiples of 3 and all x-coordinates of the left and right endpoints of the vertex segments are of the form $3i - \frac{1}{2}$ and $3j + \frac{1}{2}$, respectively, for some integers $i < j$.

The vertex gadget that replaces a vertex segment is as follows; the example shows the gadget for a segment of length 7.



The grey squares in the construction are called *connector squares*. In order to 3-colour a vertex gadget, all connector squares must receive the same colour. This colour represents the colour of the corresponding vertex in $\mathcal{G}$. Note that each edge segment passes through the center of a connector square on both vertex gadgets it connects.

The edge gadget that replaces an edge segment consists of a *basic edge gadget* plus zero or more *extension blocks*. Note that we can generate edge gadgets of vertical length $7 + 2j$ for any integer $j \geqslant 0$, by using j extension blocks. This suffices because the y-coordinates of the vertex segments are multiples of 10, and so the distance in between any two connector squares we need to connect by an edge is of the form $10k - 3$, for some integer $k \geqslant 1$. Our edge



gadget forces the connector squares of the two vertex gadgets it connects to have different colours. It is easily checked that this implies that the contact graph of the generated set of squares is 3-colourable if and only if the original graph $\mathcal{G}$ is. Moreover, the entire construction can be done in polynomial time. □

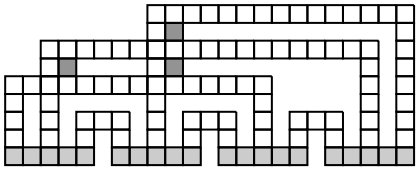
Using a similar proof we can show that 3-Colourability is NP-complete for contact graphs of discs, or of any other fixed convex and compact shape. Note that for discs (or other smooth shapes) this settles the complexity of the problem completely: contact graphs of smooth convex shapes are planar and so they are 4-colourable, and checking for 2-colourability is easy.

3 Unit Squares

If we draw the contact graph of a set of unit squares by putting vertices at the centers of the squares and drawing edges as straight segments, then the resulting drawing obviously need not be plane. The following theorem shows a stronger result, namely that contact graphs of unit squares are not planar and that in fact they can have a K_m-minor for arbitrarily large m.

Theorem 2 *For any $m \geqslant 1$, there are contact graphs of unit squares with a K_m-minor.*

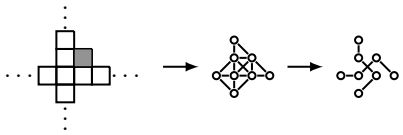
Proof. The squares we will generate to obtain a contact graph with a K_m as minor will all have integer coordinates. The following picture shows the construction for $m = 4$.



Next we explain the various components in the construction. Consider K_m. We call the nodes of the K_m *super nodes* and the edges *super edges*. For each super node u we put a block of $2m - 3$ unit squares

whose lower edges all lie on the same horizontal line. The distance between two adjacent blocks is one unit. In the figure above, the blocks are the four light grey rectangles.

For each super edge (u, v) we create a path of squares as follows. We put two vertical columns of an even number of squares—one on top of the block created for u and one on top of the block created for v—which have the same height, and we connect the topmost squares of these columns by a row of squares. We can do this such that we do not create any adjacencies between squares from different paths, except where a column of one path crosses the row of another path. Note that in this case the two paths actually share a square. Where this happens we add one more square to the top-right of the shared square—see the three dark grey squares in the picture above. These extra square allow us to obtain a minor in which all super edges are represented by disjoint paths, as the next figure shows.



By contracting the (nodes corresponding to the) square in each block to a super node and contracting the paths connecting pairs of nodes into super edges we can now obtain our K_m as a minor. Note that the construction can be done with $O(m^4)$ squares (and we can show that at least $\Omega(m^4)$ are needed). □

Despite the fact that contact graphs of unit squares are not planar, they are 4-colourable.

Theorem 3 *Any contact graph of set of unit squares is 4-colourable, and this number is tight in the worst case.*

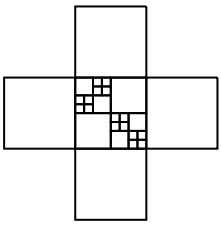
Proof. The lower bound construction is easy—just take four squares touching in a common point. For the upper bound, we divide the plane into horizontal strips of the form $(-\infty, +\infty) \times [i, i + 1)$ and assign each square to the strip containing its bottom edge. The squares assigned to a single strip can be coloured with only two colours, and by using the colour pair 1,2 for the strips with even i and 3,4 for the strips with odd i we obtain a 4-colouring. □

4 Arbitrarily-Sized Squares

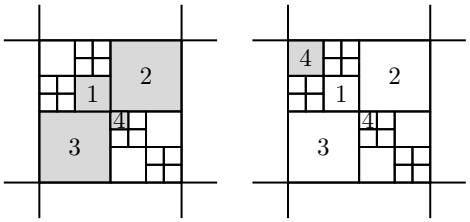
We now turn our attention to arbitrarily-sized squares.

Theorem 4 *Any contact graph of a set of squares is 6-colourable, and there are contact graphs of squares that need at least five colours.*

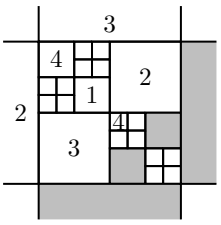
Proof. The upper bound follows from the result in the next section, where we show that even contact graphs of rectilinear polygons are 6-colourable. It remains to give an example of a set of squares that induces a contact graph that needs five colours. Eppstein *et al.* [4] already gave such an example (where the squares form a quadtree subdivision). For completeness we provide a different (and slightly smaller) example. We claim that the following graph (which is also the subgraph of a quadtree) needs at least 5 colours.



Suppose for a contradiction that the graph is 4-colourable. Then, without loss of generality, we can colour the four squares of the middle clique (consisting of four squares of different sizes) as depicted in the following picture (left). We claim that then the top left inner square has colour 4.



Indeed, if it has colour 2, none of the four squares on its right could use colour 2 and so one of these squares would need a fifth colour. Similarly, if it has colour 3, none of the four squares below it could use colour 3 and of those squares would need a fifth colour. Hence, it has to use colour 4 since it touches a square coloured with 1. Using similar arguments and simple deduction, we arrive to the following partial colouring:



Now we observe that the four gray squares form a cycle that surrounds a 4-clique. Moreover, we can easily deduce that none of the squares in the cycle can be coloured 2 or 3. Hence they have to use colour 1 and 4. But then the surrounded clique cannot use 1 or 4, a contradiction. We conclude that the graph is not 4-colourable. □

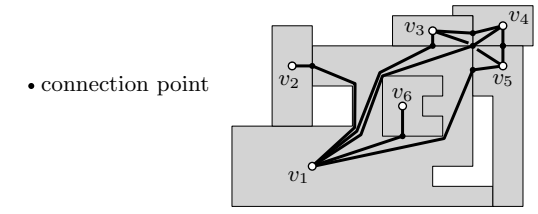
5 Rectilinear Polygons

We now turn our attention to contact squares of rectilinear polygons, where we allow the polygons to have holes. We will prove that such contact graphs are 6-colourable by showing that they are *1-planar graphs* [3], that is, graphs that can be drawn in the plane such that each edge has at most one crossing (that is, it crosses at most one other edge and this crossing then consists of a single point).

The following theorem establishes the exact relation between contact graphs of rectilinear polygons and 1-planar graphs. (We recently learned that a similar result, on the relation between 1-planar graphs and so-called 4-map graphs was already known [3]. Our proof concerns rectilinear maps and is more direct.)

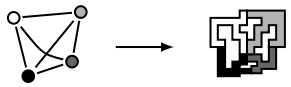
Theorem 5 *The class of contact graphs of rectilinear polygons is exactly the class of 1-plane graphs in which every pair of crossing edges is part of a K_4.*

Proof. Let $\mathcal{S} := \{P_1, \ldots, P_n\}$ be a set of interior-disjoint rectilinear polygons. To prove that the contact graph of $\mathcal{S}$ is 1-planar, we proceed as follows. First we add a point v_i in the interior of every polygon P_i, which is the embedding of the node corresponding to P_i. Next, for each pair of touching polygons P_i, P_j we pick a *connection point* $q_{ij} \in \partial P_i \cap \partial P_j$. If $\partial P_i \cap \partial P_j$ has non-zero length, we pick q_{ij} in the relative interior of $\partial P_i \cap \partial P_j$. We then embed the edge (v_i, v_j) by the union of two paths from q_{ij}: a path $\pi(q_{ij}, v_i) \subset P_i$ to v_i and a path $\pi(q_{ij}, v_j) \subset P_j$ to v_j. We do this in such a way that, for each P_i, the paths from the connection points on ∂P_i to v_i are pairwise disjoint (except at their shared endpoint v_i). This



can always be done, for example by taking a shortest-path tree rooted at v_i whose leaves are the connection points on ∂P_i. Thus an edge (v_i, v_j) can only intersect an edge (v_k, v_ℓ) when $q_{ij} = q_{k\ell}$. Since any point can be a connection point for at most two pairs of polygons, this means that in our embedding every edge intersects at most one other edge. Moreover, since all four polygons meet on the crossing point, they are part of a 4-clique.

Next we show that every 1-planar graph $\mathcal{G} = (V, E)$ with every pair of crossing edges forming a K_4 is the contact graph of a set of rectilinear polygons. Such a set can be obtained from a "pixelised" image of a 1-planar drawing of $\mathcal{G}$.



Each polygon is obtained by the vertex it represents and half of each of its edges, as shown in the picture above. If the edge is not crossing any other, we can decide arbitrarily where to divide it into the two polygons. If it crosses another edge, we cut it at the crossing point, as depicted above.

We can actually obtain a suitable set of polygons whose total number of vertices is linear in $O(|V|)$, but the proof is more complex. This bound is tight since we can construct an instance where one of the polygons has linear complexity: since for each crossing we need a corner, it suffices to have a vertex with a linear number of edges crossing other edges. □

Note that this proof works for non-rectilinear polygons as long as no five of them touch on a single point, which is always satisfied when the interior angles are strictly bigger than $2\pi/5$.

Since 1-planar graphs are 6-colourable and the figure below shows a rectilinear representation of K_6, Theorem 5 immediately implies the following.

Corollary 6 *Any contact graph of a set of rectilinear polygons is 6-colourable, and this number is tight in the worst case.*

Acknowledgment

We thank one reviewer for pointing out reference [3].

References

[1] K. Appel and W. Haken. Every map is four colourable. *Bulletin of the American Mathematical Society*, 82:711–712, 1976.

[2] O. V. Borodin. A new proof of the 6 color theorem. *Journal of Graph Theory*, 19(4):507–521, 1995.

[3] F. J. Brandenburg. On 4-map graphs and 1-planar graphs and their recognition problem. *CoRR*, abs/1509.03447, 2015.

[4] D. Eppstein, M. W. Bern, and B. L. Hutchings. Algorithms for coloring quadtrees. *Algorithmica*, 32(1):87–94, 2002.

[5] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976.

[6] P. Koebe. Kontaktprobleme der konformen abbildung. *Ber. Verh. Sächs. Akademier der Wissenschaften Leipzig, Math.-Phys. Klasse 88*, pages 141–164, 1936.

[7] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry*, 1:343–353, 1986.

Connected Dominating Set in Unit-Disk Graphs is W[1]-hard*

Mark de Berg† Hans Bodlaender‡† Sándor Kisfaludi-Bak†

Abstract

We prove that connected dominating set is W[1]-hard for unit-disk graphs.

1 Introduction

Wireless networks give rise to a host of interesting algorithmic problems. In the traditional model of a wireless network each node u corresponds to a disk D_u in the plane, whose radius equals the transmission range of u. Thus u can send a message to another node v if and only if $v \in D_u$. If each node has the same transmission range and we shrink each disk by a factor two, this condition is equivalent to requiring that the (shrunk) disks D_u and D_v intersect. Thus the communication graph is the intersection graph of a collection of congruent disks or, in other words, a *unit-disk graph (UDG)*. Because of their relation to wireless networks, UDGs have been studied extensively.

Let $\mathcal{D}$ be a set of disks in the plane, and let $\mathcal{G}_\mathcal{D} = (\mathcal{D}, E)$ be the UDG induced by $\mathcal{D}$. A *broadcast tree* is a rooted spanning tree for $\mathcal{G}$. To send a message from the root of the broadcast tree to all other nodes, each internal node of the tree has to send the message to its children. Hence, the cost of broadcasting is related to the number of internal nodes in the broadcast tree. A cheapest broadcast tree thus corresponds to a minimum-size *connected dominating set* on $\mathcal{G}_\mathcal{D}$, that is, a minimum-size subset $\Delta \subset \mathcal{D}$ such that the subgraph induced by Δ is connected and each node in $\mathcal{G}_\mathcal{D}$ is either in Δ or it is a neighbor of a node in Δ. Thus we are interested in the following problem: given a set $\mathcal{D}$ of n disks and a parameter k, does $\mathcal{G}_\mathcal{D}$ admit a connected dominating set of size at most k?

In the following we denote the dominating-set problem by DS, the connected dominating-set problem by CDS, and we denote these problems on UDGs by DS-UDG and CDS-UDG, respectively. It is well known that DS and CDS are NP-hard, even for planar graphs [5]. DS-UDG and CDS-UDG are also NP-hard [6, 8]. In this paper we are interested in the parameterized complexity [4] of these problems, with k being the parameter.

*This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003.

†Department of Computer Science, TU Eindhoven. mdberg@win.tue.nl, s.kisfaludi.bak@tue.nl

‡Department of Information and Computing Sciences, Utrecht University, h.l.bodlaender@uu.nl

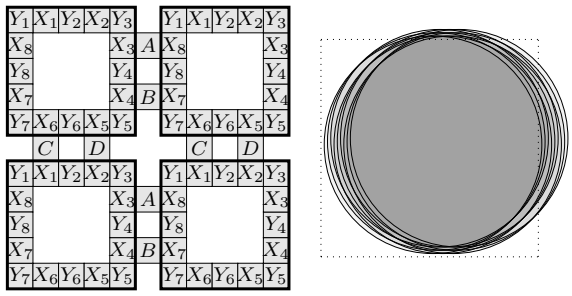


Figure 1: Left: the construction of Marx for $k = 2$. Right: example of disks inside a single block (X_2).

For general graphs DS and CDS are well-known W[2]-complete problems, but for planar graphs both problems are fixed-parameter tractable [1, 3]. The question is what happens for unit-disk graphs, which are in between general graphs and planar graphs. Marx [7] showed that DS-UDG is W[1]-hard; in this paper we extend his construction to show that CDS-UDG is W[1]-hard as well. The membership in W[1] remains open for both DS-UDG and CDS-UDG.

2 The construction by Marx for DS-UDG

Our W[1]-hardness proof for CDS-UDG has the same global structure as the W[1]-hardness proof of Marx [7] for DS-UDG. Hence, we first describe his proof. He uses a reduction from Grid Tiling [2] (although Marx does not explicitly state it this way). In a grid-tiling problem we are given an integer k, an integer n, and a collection $\mathcal{S}$ of k^2 non-empty sets $S_{a,b} \subseteq [n] \times [n]$ $(1 \leqslant a, b \leqslant k)$, and the goal is to select an element $s_{a,b} \in S_{a,b}$ for each $1 \leqslant a, b \leqslant k$ such that

- If $s_{a,b} = (x, y)$ and $s_{a+1,b} = (x', y')$, then $x = x'$.
- If $s_{a,b} = (x, y)$ and $s_{a,b+1} = (x', x')$, then $y = y'$.

One can picture these sets in a $k \times k$ matrix: in each cell (a, b), we need to select a representative from the set $S_{a,b}$ so that the representatives selected from horizontally neighboring cells agree in the first coordinate, and representatives from vertically neighboring sets agree in the second coordinate.

The reduction places k^2 gadgets, one for each $S_{a,b}$. A gadget contains sixteen blocks, labeled $X_1, Y_1, X_2, Y_2, \ldots, X_8, Y_8$, that are arranged in a grid.

Initially, each block X_ℓ contains n^2 disks, denoted by $X_\ell(1), \ldots, X_\ell(n^2)$ and each block Y_ℓ contains n^2+1 disks denoted by $Y_\ell(0), \ldots, Y_\ell(n^2)$. The argument j of $X_\ell(j)$ can be thought of as a pair (x, y) with $1 \leqslant x, y \leqslant n$ for which $f(x, y) = (x-1)n+y = j$. Let $f^{-1}(j) = (\iota_1(j), \iota_2(j)) = (1 + \lfloor j/n \rfloor, 1 + (j \mod n))$.

For the final construction, in each gadget at position (a, b), delete all disks $X_\ell(j)$ for each $\ell = 1, \ldots, 8$ and $(\iota_1(j), \iota_2(j)) \notin S_{a,b}$. This deletion ensures that the gadgets represent the corresponding set $S_{a,b}$. (The disks of a minimum dominating set in the gadget (a, b) will signify a specific choice $s_{a,b} = (x, y)$.)

Moreover, there are special connector blocks (denoted by A, B, C and D) between neighboring gadgets, each of them containing $n+1$ disks. A picture of the construction for $k = 3$ can be seen in Figure 1, where each block is represented by a square.

In every block, the place of each disk center is defined with regard to the midpoint of the block, (r, s). The center of each circle is of the form $(r+\alpha\epsilon, s+\beta\epsilon)$ where r, s, α and β are integers, and $\epsilon > 0$ a small constant. We say that the *offset* of the disk centered at $(r + \alpha\epsilon, s + \beta\epsilon)$ is (α, β). Note that $|\alpha|, |\beta| \leqslant n$, and $\epsilon < n^{-2}$, so the disks in a block all intersect each other. The disks of a block can be thought of as slightly shifted versions of the inscribed disk of the square in Figure 1. The exact offsets for each disk are defined in [7]. We only describe the important properties. First, two disks can intersect only if they are in the same or in neighboring blocks. Consequently, one needs at least 8 disks to dominate a gadget. The second important property is that disk $X_\ell(j)$ dominates exactly $Y_\ell(j), \ldots, Y_\ell(n^2)$ from the "previous" block Y_ℓ, and $Y_{\ell+1}(0), \ldots, Y_{\ell+1}(j-1)$ from the "next" block $Y_{\ell+1}$). This property can be used to prove the following key lemma.

Lemma 1 *Assume that a gadget is part of an instance such that none of the blocks Y_i are intersected by disks outside the gadget. If there is a dominating set Δ of the instance that contains exactly $8k^2$ disks, then there is a* canonical *dominating set Δ' with $|\Delta'| = |\Delta|$, such that for each gadget G, there is an integer $1 \leqslant j^G \leqslant n$ such that Δ' contains exactly the disks $X_1(j^G), \ldots, X_8(j^G)$ from G.*

In the gadget $G_{a,b}$, the value j defined in the above lemma represents the choice of $s_{a,b} = (\iota_1(j), \iota_2(j))$ in the grid tiling problem. Our deletion of certain disks in X-blocks ensures that $(\iota_1(j), \iota_2(j)) \in S_{a,b}$. Finally, in order to get a feasible grid tiling, gadgets in the same row must agree on the first coordinate, and gadgets in the same column must agree on the second coordinate. This depends on the following lemma.

Lemma 2 *Let Δ be a canonical dominating set. For horizontally neighboring gadgets G and H representing j_G and j_H, the disks of the connector block A are dominated if and only if $\iota_1(j_G) \leqslant \iota_1(j_H)$; the disks of B are dominated if and only if $\iota_1(j_G) \geqslant \iota_1(j_H)$. Similarly, for vertically neighboring blocks G' and H', the disks of block C are dominated if and only if $\iota_2(j_{G'}) \leqslant \iota_2(j_{H'})$; the disks of D are dominated if and only if $\iota_2(j_{G'}) \geqslant \iota_2(j_{H'})$.*

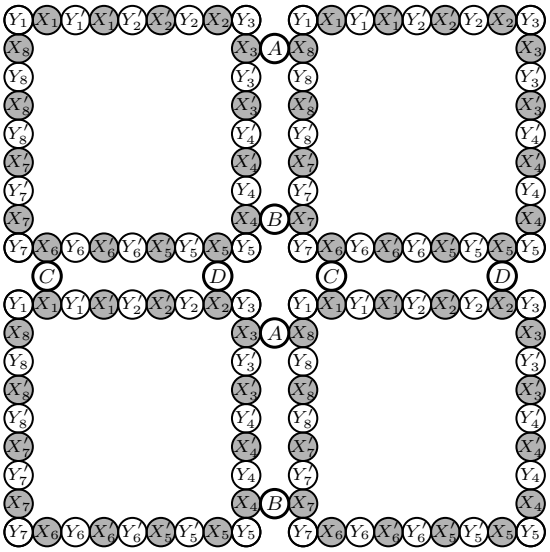


Figure 2: Connecting neighbouring blocks

With the above lemmas, it is easy to see how the reduction works. A feasible grid tiling defines a dominating set of size $8k^2$: in gadget $G_{a,b}$, the dominating disks are $X_\ell(f(s_{a,b}))$, $\ell = 1, \ldots, 8$. On the other hand, if there is a dominating set of size $8k^2$, then there is a canonical dominating set of the same size that defines a feasible grid tiling.

3 New construction for CDS-UDG

To extend the construction to CDS-UDG, we want to make sure that minimum-size dominating set is connected. This requires two things. First, we must add new disks "inside" the gadgets — that is, in the empty space surrounded by the X and Y-blocks — such that a canonical minimum dominating set includes some new disks that connect the chosen $X_\ell(j)$ disks without interfering with disks in the Y-blocks. Second, we need to connect all the different gadgets. This time in addition to avoiding the Y-blocks, we also need to avoid interfering with the connector blocks.

In order to have enough space, our gadgets contain 32 blocks instead of 16. The offsets of disks inside the blocks are not modified: we use the same building blocks. Figure 2 shows how we arrange these blocks, and depicts the connector-block placement.

The analogue of Lemma 1 and Lemma 2 are true here; we have a construction that could be used to

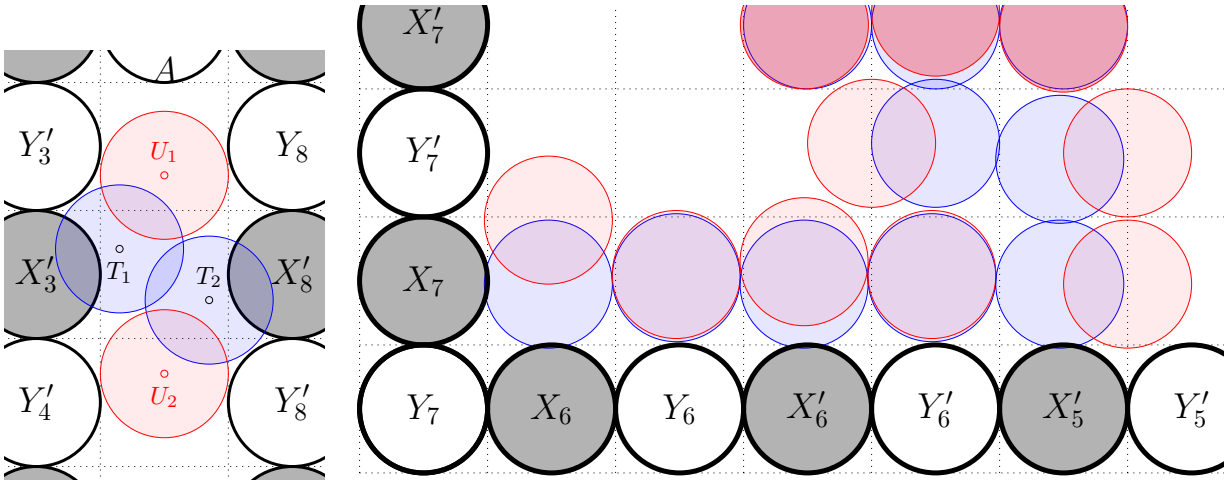


Figure 3: Left: connecting horizontally; right: connecting one side to the middle.

prove the W[1]-hardness of DS-UDG, with canonical sets of size $16k^2$, that contain one disk from each X-block and X'-block. We extend this construction so that we have canonical dominating sets that span a connected subgraph.

The disks we add are always in pairs. One of these disks (the *parent*) "connects" some other disks, or more specifically, the set of parents together with one arbitrary disk from each X and X' block is a connected subgraph of our construction. The other disk (called *leaf*) only intersects its parent and it is disjoint from all other disks. Let Δ be a dominating set. In Δ, at least one of the parent or the leaf has to be included so that the leaf is dominated. Hence, we can assume that a minimum size dominating set contains all parent disks, which (as we will ensure) form a connected set.

The most important property of the blocks that we use is that for a small enough value ϵ, the boundaries of the disks in a block all lie inside a small width annulus - for this reason, the blocks in our pictures are depicted with thick boundary disks. In order for a parent disk p to intersect every disk in a block it is sufficient if the boundary of p crosses this annulus.

We are going to add 72 extra disks to every gadget, and 4 "connector" disks between every pair of horizontally or vertically neighboring gadgets, resulting in canonical dominating sets of size $16k^2+36k^2+4k(k-1) = 56k^2-4k$ (Note that only the parent disks are included in the canonical set). In other words, the new construction has a connected dominating set of size $56k^2 - 4k$ if and only if there is a feasible grid tiling. Due to length constraints we will not be able to list the coordinates of these disks and prove all the intersections/disjointness that is required. These details will be available in the final version of this paper.

Connecting neighboring gadgets. For a pair of horizontally neighboring gadgets, we add two pairs of disks that connect X'_3 from the left gadget to X'_8 in the right gadget. This arrangement is depicted on the left of Figure 3. The parent disk with center T_1 intersects every disk in the block X'_3 of the left gadget, and the other parent intersects every disk in the block X'_8. The two leaf disks (red disks in the figure) only intersect their parent. We use a rotated version of these 4 disks for vertical connections, where the parents connect X'_5 from the upper gadget and X'_2 from the lower gadget.

Disks inside gadgets. We begin by adding 8 disk pairs to the center. The parents are arranged in a square, touching the neighbors, and the leafs are placed so that it is possible to connect from the outside on each side. See the middle of Figure 4 for a picture: the corresponding leaf disks have a darker shade of red.

In order to connect the X-blocks, we need to connect the blocks of each side to the central disks. For this purpose, we are going to use a zigzag pattern of disks. The first parent disk intersect all disks in X_6 and X_7 (i.e., it crosses both annuli), the second parent is above the block Y_6, but it is disjoint from it. The next with center P_3 intersects all disks in X'_6 , and the disk around P_4 is disjoint from the disks in Y'_6. Finally, the disk around P_5 intersects all disks in X'_5. The leafs follow a more complicated pattern. This pattern is depicted on the right side of Figure 3. Our final gadget can be attained by rotating the above seven disk pairs around the center $(8, 8)$ by $90, 180$ and 270 degrees: see Figure 4. We added the spanned edges of a canonical dominating set to this picture. This concludes the proof of our main theorem.

Theorem 3 *The* CDS-UDG *problem is W[1]-hard.*

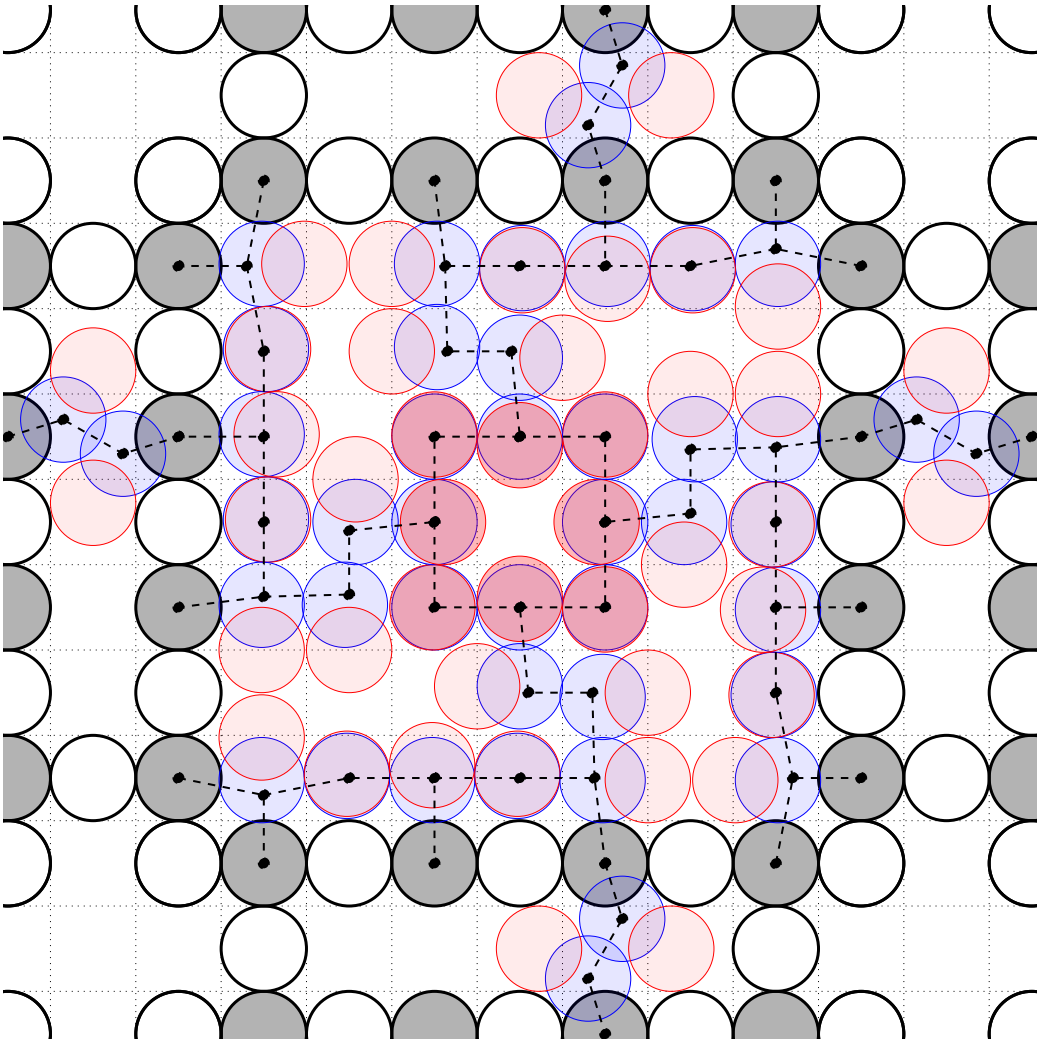


Figure 4: A gadget in the final construction. The dashed lines are spanned edges of a canonical dominating set.

References

[1] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for DOMINATING SET and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[2] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[3] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[4] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[6] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[7] D. Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland*, pages 154–165, 2006.

[8] S. Masuyama, T. Ibaraki, and T. Hasegawa. The computational complexity of the m-center problems on the plane. *IEICE TRANSACTIONS (1976-1990)*, 64(2):57–64, 1981.

Flip Distance to a Non-crossing Perfect Matching

Édouard Bonnet * Tillmann Miltzow †

Abstract

A *perfect straight-line matching* M on a finite set P of points in the plane is a set of segments such that each point in P is an endpoint of exactly one segment. M is *non-crossing* if no two segments in M cross each other. Given a perfect straight-line matching M with at least one crossing, we can remove this crossing by a flip operation. The flip operation removes two crossing segments on a point set Q and adds two non-crossing segments to attain a new perfect matching M'. It is well known that after a finite number of flips, a non-crossing matching is attained and no further flip is possible. However, prior to this work, no non-trivial upper bound on the number of flips was known. If $g(n)$ (resp. $k(n)$) is the maximum length of the longest (resp. shortest) sequence of flips starting from any matching of size n, we show that $g(n) = O(n^3)$ and $g(n) = \Omega(n^2)$ (resp. $k(n) = O(n^2)$ and $k(n) = \Omega(n)$).

Van Leeuwen and Schoone showed with the same argument and the same definition of flip how to transform a Hamilton cycle to a non-crossing Hamilton cycle on a set of n points within $O(n^3)$ flips [17]. Therefore, we do not consider the main result (our upper bound on $g(n)$) as a new contribution, because the used technique is exactly the same.

We want to use these proceedings to draw attention again on this old problem and hope to stimulate research that will close the gap between the upper and lower bound.

1 Introduction

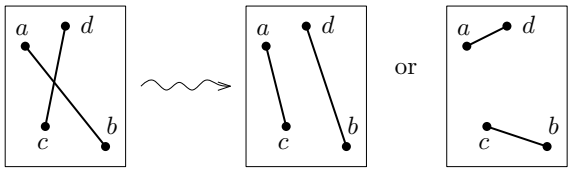


Figure 1: Two crossing segments are replaced by two non-crossing segments. There are two ways to flip.

Given $2n$ points in the plane in general position (no three points on a line), we define a *perfect straight-line non-crossing matching* as a set of n segments such that each point is incident to exactly one segment and no two segments intersect. Given $2n$ points in the plane, it is well-known that a perfect straight-line non-crossing matching always exists. One elegant argument to see this is to start with any perfect straight-line matching, potentially self-intersecting, and remove any crossing by a flip (see Figure 1). Although the total number of crossings might increase (see Figure 5), the sum of the length of all the segments decreases (see Figure 2). Thus, the process will eventually end with a perfect non-crossing matching.

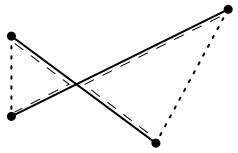


Figure 2: The two new edges (dotted) are shorter than the old edges (solid) since the dashed part to the left (resp. to the right) of the crossing is longer than the dotted segment on the left (resp. on the right).

A simpler argument is to take the first two points with lowest x-coordinate and connect them with a segment and continue with the remaining points by induction. Contrary to the first argument, this does not carry over to the bichromatic setting. In the bichromatic setting, we are given n red and n blue points and a *bichromatic matching* is a matching as above with the additional property that only segments linking points of different colors are allowed.

Motivated by this old folklore result, we investigate the question on the maximum and minimum number of flips that are necessary and sufficient to reach a straight-line non-crossing matching.

1.1 Preliminaries

From here on, P always denotes a set of $2n$ points in the plane and M a perfect straight-line matching on P. Given two points a and b we denote by $\mathrm{seg}(a, b)$ the segment with endpoints a and b. Given a perfect straight-line matching M with at least one crossing, we can remove this crossing by a flip operation. The flip operation removes any two crossing segments

*Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), edouard.bonnet@lamsade.dauphine.fr

†Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), t.miltzow@gmail.com

$\text{seg}(a,b)$ and $\text{seg}(c,d)$ on a point set $Q = \{a,b,c,d\}$ and adds two non-crossing segments ($\text{seg}(a,c)$ and $\text{seg}(b,d)$) or ($\text{seg}(a,d)$ and $\text{seg}(b,c)$) to attain a new matching M'. Matching M is a *successor* of matching M' if we can construct M from M' by a single flip. We say that $\mathcal{M} = (M_0, \ldots, M_k)$ is a valid sequence of matchings, if each matching M_{i+1} is a successor of M_i and M_k is non-crossing. The number k denotes the length of $\mathcal{M}$. Given a set P of $2n$ points in the plane, we define:

$$f(M) = \max\{\, k : \exists \mathcal{M} \text{ of length } k \text{ with } M = M_0 \,\};$$

$$h(M) = \min\{\, k : \exists \mathcal{M} \text{ of length } k \text{ with } M = M_0 \,\}.$$

Consequently functions $g(n)$ and $k(n)$ are defined as:

$$g(n) = \max\{\, f(M) : M \text{ is a matching on } 2n \text{ points} \,\};$$

$$k(n) = \max\{\, h(M) : M \text{ is a matching on } 2n \text{ points} \,\}.$$

1.2 Results

We establish the following result:

Theorem 1 ([17]) *Let n be a large enough natural number. Then it holds:*

$$\frac{n^2 - n}{2} = \binom{n}{2} \leq g(n) \leq n^3.$$

This result immediately carries over to bichromatic matchings. We conjecture that $g(n) = \Theta(n^2)$.

Theorem 2 *Let n be a large enough natural number. Then it holds:*

$$n - 1 \leq k(n) \leq \frac{n^2}{2},$$

for some constants C.

Our proof of Theorem 2 does *not* carry over to the bichromatic case. However, we will see that the upper bound further holds if the crossing to flip is imposed at each step by an adversary and we may only choose which of the two flips (see Figure 1) we perform.

1.3 Related Work

The most relevant work is by van Leeuwen and Schoone, who showed the same upper bound on $g(n)$ for Hamilton cycles instead of matchings [17]. The study of Hamilton cycles is motivated as a post-processing step for algorithms that find a traveling salesman tour on a set of points. They show that they can improve the solution to be non-crossing in $O(n^3)$ steps.

For points in convex position, Oda and Watanabe established linear upper and lower bounds on $g(n)$, again on Hamilton cycles instead of matchings [15].

The combinatorial work on flip graphs of geometric structures is fairly large. See the survey by Bose and Hurtado [11] for an overview and some motivations.

Matchings, triangulations and spanning trees are commonly studied in recent work [1–9, 12, 13, 16]. Particularly interesting are triangulations of points that are in convex positions as they correspond to Catalan structures. Another interesting application comes from Lawson flips, which can be used to reach the Delaunay triangulation in $O(n^2)$ flips [14]. This can be used for the reverse search technique to enumerate triangulations [10].

2 Lower Bounds

We start with the lower bound for Theorem 1 and 2. Let ℓ and ℓ' be two parallel horizontal lines and let P be a set of $2n$ points, n of which are on ℓ and ℓ' respectively. In the following, we consider only matchings that connect points from ℓ to ℓ' (see Figure 3). Every

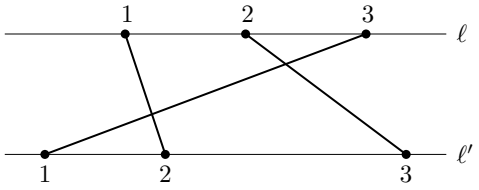


Figure 3: Matching corresponding to cycle (123).

such matching M can be interpreted as a permutation π_M and M is crossing free if and only if π_M is the identity. We can always do flips that correspond to an elementary step in bubble sort. Bubble sort on permutation π needs as many steps as the number of inversions of π. And, the number of inversions is at most $\binom{n}{2}$. A small perturbation of the point set ensures general position.

For the lower bound of Theorem 2, we define a *Schoone matching* on $2n$ points in convex position denoted by $p_1, p_2, \ldots, p_{2n}$ in counterclockwise order as follows. The points p_1 is linked to p_{n+1} and for each $i \in [2,n]$, p_i to p_{2n+2-i} (see Figure 4). Observe that

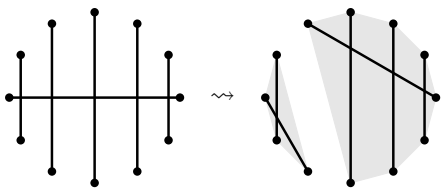


Figure 4: An initial configuration guaranteeing the lower bound of Theorem 2, and a possible flip.

a Schoone matching decomposes into two Schoone matchings after any flip. By this the total number of

crossings decreases by exactly one. This implies by induction that $h(M)$ equals the number of crossings minus one, for all Schoone matchings. Thus $n-1 \leq k(n)$.

3 Upper Bounds

Before we prove the upper bound, observe in Figure 5 that the number of crossings might increase after a flip. It is also possible that a segment that has disappeared after a flip reappear after some more flips (see Figure 6). These two observations suggest that there is no straightforward way of getting a good upper bound.

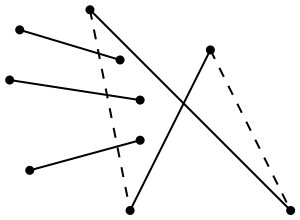


Figure 5: After the depicted flip, the number of crossings goes from 1 to 3.

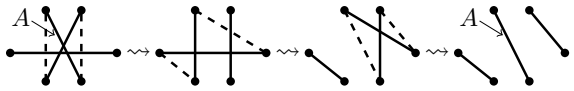


Figure 6: Segment A disappears and reappears.

For the upper bound of Theorem 1, we define a potential function $\Phi_{\mathcal{L}}(M)$ that depends on a well-chosen set of lines $\mathcal{L}$. We show that $\Phi_{\mathcal{L}}(M) \leq 4n^3$ and that $\Phi_{\mathcal{L}}$ decreases by at least four after any flip. The potential function $\Phi_{\mathcal{L}}(M)$ is defined as the number of intersections between a line of $\mathcal{L}$ and a segment of M. We define $\mathcal{L}$ as follows. Given two points $p, q \in P$ let ℓ be the supporting line of p and q. We add to $\mathcal{L}$ the two lines slightly above and below ℓ (see Figure 7).

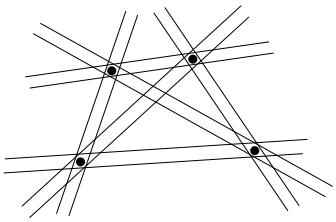


Figure 7: Construction of $\mathcal{L}$.

It holds that $|\mathcal{L}| = 2\binom{2n}{2} \leq 4n^2$. As any line and segment can cross at most once it follows $\Phi_{\mathcal{L}}(M) \leq |\mathcal{L}| \cdot |M| = 4n^3$. It remains to show that the number of segment-line intersections decreases by at least four in any flip. Consider two crossing segments A and B on points $Q = \{ p_1, \ldots, p_4 \}$ as in Figure 8. Note that there are only three combinatorial types of lines intersecting the convex hull of Q. Either a line separates p_1 and p_2 from p_3 and p_4 as ℓ_1; a line separates p_1 and p_4 from p_2 and p_3 as ℓ_2; or a line separates one point from the other three as ℓ_3. For every type of lines the number of intersections does not increase after flipping A and B. It is also easy to see that the number of intersections decreases by two for lines of type ℓ_1 or ℓ_2 after flipping A and B. By definition of $\mathcal{L}$, there exists for every crossing of two segments at least two lines of type ℓ_1 and at least two lines of type ℓ_2. Thus $\Phi_{\mathcal{L}}(M)$ decreases by at least four as claimed.

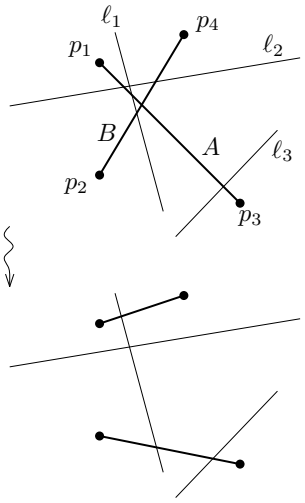


Figure 8: Flipping A and B yields fewer segment-line intersections.

For the upper bound of Theorem 2, we define a different set of lines $\mathcal{K}$,which contains one vertical line between any two consecutive points ordered in x-direction, see Figure 9. It follows that $\Phi_{\mathcal{K}}(M) \leq n^2$ since $|\mathcal{K}| = n - 1$. We have to show that $\Phi_{\mathcal{K}}$ decreases by at least two after each flip. Let A and B be two crossing segments on the points p_1, p_2, p_3, p_4 ordered by x-coordinate. Then we replace A and B by $\text{seg}(p_1, p_2)$ and $\text{seg}(p_3, p_4)$, see Figure 10. It is clear that at least one line ℓ between p_2 and p_3 is not crossed after the flip and was crossed twice before the flip.

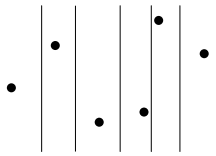


Figure 9: The set $\mathcal{K}$.

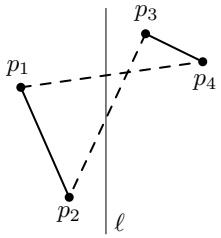


Figure 10: The number of crossings between ℓ and the segments of the matchings decreases by 2.

Acknowledgments

We want to thank anonymous reviewers for the hint to the paper of van Leeuwen and Schoone [17] and a hint to the paper by Oda and Watanabe [15]. Both authors are supported by the ERC grant PARAMTIGHT: "Parameterized complexity and the search for tight complexity results", no. 280152. The second author thanks the organizers and participants of the Emléktábla Workshop in the Summer 2015 for a nice event and interesting discussions respectively. (There are too many to list names and we do not want to forget someone.)

References

[1] Oswin Aichholzer, Andrei Asinowski, and Tillmann Miltzow. Disjoint compatibility graph of non-crossing matchings of points in convex position. *Electr. J. Comb.*, 22(1):P1.65, 2015.

[2] Oswin Aichholzer, Franz Aurenhammer, Clemens Huemer, and Hannes Krasser. Transforming spanning trees and pseudo-triangulations. *Inf. Process. Lett.*, 97(1):19–22, 2006.

[3] Oswin Aichholzer, Sergey Bereg, Adrian Dumitrescu, Alfredo García Olaverri, Clemens Huemer, Ferran Hurtado, Mikio Kano, Alberto Márquez, David Rappaport, Shakhar Smorodinsky, Diane L. Souvaine, Jorge Urrutia, and David R. Wood. Compatible geometric matchings. *Comput. Geom.*, 42(6-7):617–626, 2009.

[4] Oswin Aichholzer, Thomas Hackl, David Orden, Alexander Pilz, Maria Saumell, and Birgit Vogtenhuber. Flips in combinatorial pointed pseudo-triangulations with face degree at most four. *Int. J. Comput. Geometry Appl.*, 24(3):197–224, 2014.

[5] Oswin Aichholzer, Thomas Hackl, David Orden, Pedro Ramos, Günter Rote, André Schulz, and Bettina Speckmann. Flip graphs of bounded degree triangulations. *Graphs and Combinatorics*, 29(6):1577–1593, 2013.

[6] 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.

[7] Greg Aloupis, Luis Barba, Stefan Langerman, and Diane L. Souvaine. Bichromatic compatible matchings. *Comput. Geom.*, 48(8):622–633, 2015.

[8] Andrei Asinowski, Tillmann Miltzow, and Günter Rote. Quasi-parallel segments and characterization of unique bichromatic matchings. *Journal on Computational Geometry*, 6(1):185–219, 2015.

[9] Andrei Asinowski and Günter Rote. Point sets with many non-crossing matchings. *CoRR*, abs/1502.04925, 2015.

[10] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.

[11] Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Comput. Geom.*, 42(1):60–80, 2009.

[12] Michael E. Houle, Ferran Hurtado, Marc Noy, and Eduardo Rivera-Campo. Graphs of triangulations and perfect matchings. *Graphs and Combinatorics*, 21(3):325–331, 2005.

[13] Mashhood Ishaque, Diane L Souvaine, and Csaba D. Tóth. Disjoint compatible geometric matchings. *Discrete & Computational Geometry*, 49(1):89–131, 2013.

[14] Charles L Lawson. Properties of n-dimensional triangulations. *Computer Aided Geometric Design*, 3(4):231–246, 1986.

[15] Yoshiaki Oda and Mamoru Watanabe. The number of flips required to obtain non-crossing convex cycles. In *Computational Geometry and Graph Theory*, pages 155–165. Springer, 2008.

[16] Alfredo García Olaverri, Clemens Huemer, Ferran Hurtado, and Javier Tejel. Compatible spanning trees. *Comput. Geom.*, 47(5):563–584, 2014.

[17] Jan van Leeuwen, Anneke A. Schoone. *Untangling a traveling salesman tour in the plane.* Rijksuniversiteit. Vakgroep Informatica, 1980.

Coloring and $L(2,1)$-labeling of unit disk intersection graphs

Konstanty Junosza-Szaniawski*† Pawe Rzążewski*‡ Joanna Sokół* Krzysztof Węsek*

Abstract

In this paper we give a family of on-line algorithms for the classical coloring problem and the $L(2,1)$-labeling of unit disc intersection graphs. Our algorithms make use of a geometric representation of such graphs and are inspired by an algorithm of Fiala *et al.*, but have better competitive ratios. The improvement comes from an application of a fractional and a b-fold coloring of the plane. Moreover, we give an off-line algorithm improving the bound of the $L(2,1)$-span of unit disk intersection graphs in terms of the maximum degree.

1 Introduction

Intersection graphs of families of geometric objects attracted much attention of researches both for their theoretical properties and practical applications (c.f. McKee and McMorris [10]). For example intersection graphs of families of discs, and in particular discs of unit diameter (called *unit disk intersection graphs*), play a crucial role in modeling radio networks. Apart from the classical coloring, other labeling schemes such as T-coloring and distance-constrained labeling of such graphs are applied to frequency assignment in radio networks [9, 13].

In this paper we consider the classical coloring and the $L(2,1)$-labeling. The latter asks for a vertex labeling with non-negative integers, such that adjacent vertices get labels that differ by at least two, and vertices at distance two get different labels. The *span* of an $L(2,1)$-labeling is the maximum label used. The $L(2,1)$-*span* of a graph G, denoted by $\lambda(G)$, is the minimum span of an $L(2,1)$-labeling of G (note that the number of available labels is $\lambda(G)+1$, but some may not be used).

We say that a graph coloring algorithm is *on-line* if the input graph is not known a priori, but is given vertex by vertex (with all edges adjacent to already revealed vertices). Each vertex is colored at the moment when it is presented and its color cannot be changed later. On the other hand, *off-line* coloring algorithms know the whole graph before they start assigning colors. The on-line coloring can be much harder than off-line coloring, even for paths. For an off-line coloring algorithm (off-line $L(2,1)$-labeling algorithm, resp.), by the *approximation ratio* we mean the worst-case ratio of the number of colors used by this algorithm (the largest label used by this algorithm, resp.) to the chromatic number of the graph ($\lambda(G)$, respectively). For on-line algorithms, the same value is called the *competitive ratio*.

A unit disk intersection graph G can be colored off-line in polynomial time with $3\omega(G)$ colors [12] (where $\omega(G)$ denotes the size of a maximum clique) and on-line with $5\omega(G)$ colors [11, 12]. Fiala *et al.* [3] presented an on-line algorithm that finds an $L(2,1)$-labeling of a unit disk intersection graph with span not exceeding $25\omega(G)$. The algorithm is based on a special pre-coloring of the plane, that resembles colorings studied by Exoo [2], inspired by the classical Hadwiger-Nelson problem [8]. Our main result are on-line algorithms for the coloring and the $L(2,1)$-labeling of unit disc intersection graphs with better competitive ratios than previous algorithms. They are inspired by [3], although a b-fold coloring of the plane (see [7]) is used instead of a classical coloring. In particular, in the case of using 1-fold coloring we obtain the algorithm from [3]. Our algorithm colors (in the classical sense) unit disc intersection graphs with large maximum clique, using less than $5\omega(G)$ colors and hence it is the best currently known approximation on-line coloring algorithm for such graphs. For $L(2,1)$-labeling, in the case of 1-fold coloring of the plane, our algorithm gives a labeling with span not exceeding $20\omega(G)$. Using b-fold coloring for $b>1$ we obtain even better results.

For general graphs, Griggs and Yeh proved that $\lambda(G) \leq \Delta(G)^2 + 2\Delta(G)$ and conjectured that $\lambda(G) \leq \Delta(G)^2$. Shao *et al.*[14] showed $\lambda(G) \leq \frac{4}{5}\Delta(G)^2 + 2\Delta(G)$ if $G \in UDG$. Actually, they gave an on-line algorithm that finds an $L(2,1)$-labeling of G with span at most $\frac{4}{5}\Delta(G)^2 + 2\Delta(G)$. We managed to improve this bound to $\frac{3}{4}\Delta^2 + 3(\Delta-1)$, in the off-line case. Moreover, we show that the algorithm from [3] implies the bound $18\Delta+18$, which is better for $\Delta \geq 22$.

Throughout the paper we always assume that the input unit disk intersection graph is given along with its geometric representation. In practical application for mobile Wi-Fi routers representation can be found

*Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland. E-mail: {k.szaniawski, p.rzazewski, j.sokol, k.wesek}@mini.pw.edu.pl

†Supported by the National Centre for Research and Development, grant No. PBS2/B3/24/2014, application No. 208921.

‡Institute of Computer Science and Control, Hungarian Academy of Sciences, Hungary. Supported by ERC Starting Grant PARAMTIGHT (No. 280152).

with methods from [5].

2 Preliminaries

For an integer n, we define $[n] := \{1, \ldots, n\}$. A function $c\colon V \to [k]$ is a k-coloring of $G = (V, E)$ if for any $xy \in E$ holds $c(x) \neq c(y)$. By $d(u, v)$ we denote the number of edges on the shortest u-v–path in G.

For a sequence of unit discs in the plane $(D_i)_{i\in[n]}$ we define its intersection graph by $G((D_i)_{i\in[n]}) = (\{v_i : i \in [n]\}, E)$, where v_i is the center of D_i for every $i \in [n]$ and $v_iv_j \in E$ iff $D_{v_i} \cap D_{v_j} \neq \emptyset$. Notice that $v_iv_j \in E$ if and only if the Euclidean distance between v_i and v_j, denoted by $\mathrm{dist}(v_1, v_2)$, is at most one. By *UDG* we mean the class of graphs that admit a representation by intersecting unit disks.

For a minimization on-line algorithm alg, by $\mathrm{cr}(\mathrm{alg})$ we denote its *competitive radio*, which is the supremum of $\frac{\mathrm{alg}(G)}{\mathrm{opt}(G)}$ over all instances G, where $\mathrm{alg}(G)$ is the value of the solution given by the algorithm for instance G and $\mathrm{opt}(G)$ is the optimal solution for instance G. For the classical coloring we use fact that any coloring requires at least $\omega(G)$ colors, where $\omega(G)$ denotes the size of the largest clique of G. By $\mathcal{G}_\omega$ we denote the class of graphs with largest clique of size at least ω and by $\mathrm{cr}(\mathrm{alg}(\mathcal{G}_\omega))$ we denote the supremum of $\frac{\mathrm{alg}(G)}{\mathrm{opt}(G)}$ over all graphs $G \in \mathcal{G}_\omega$.

A *tiling* is a partition of the plane into convex polygons with partially removed boundary, called *tiles*, such that every two points from one tile are at distance less than one. If we have b tilings, then by a *subtile* we mean a non-empty intersection of b tiles, one from each tiling. We will use a hexagon as a tile and hexagon tiling, just as Fiala *et al.* [3].

A function $\varphi : \mathbb{R}^2 \to [k]$ is called a *coloring of the plane with the color set* $[k]$ if for any two points $p_1, p_2 \in \mathbb{R}^2$ with $\mathrm{dist}(p_1, p_2) = 1$ holds $\varphi(p_1) \neq \varphi(p_2)$.

Definition 1 *A function $\varphi = (\varphi_1, \ldots, \varphi_b)$ where $\varphi_i : \mathbb{R}^2 \to [k]$ for $i \in [b]$ is called a b-*fold coloring of the plane with color set $[k]$ *if*

- *for any point $p \in \mathbb{R}^2$ and $i, j \in [b]$, if $i \neq j$, then $\varphi_i(p) \neq \varphi_j(p)$,*
- *for any two points $p_1, p_2 \in \mathbb{R}^2$ with $\mathrm{dist}(p_1, p_2) = 1$ and $i, j \in [b]$ holds $\varphi_i(p_1) \neq \varphi_j(p_2)$.*

*The function φ_i for $i \in [b]$ is called an i-*th layer *of φ.*

Notice that a coloring of the plane is a 1-fold coloring of the plane. A coloring of a plane φ is called tiling-based if there exists a tiling such that each tile is monochromatic and adjacent tiles have different colors. A b-fold coloring of a plane $\varphi = (\varphi_1, \ldots, \varphi_b)$ is called tiling-based if for every $i \in [b]$ coloring φ_i is tiling-based.

For technical reasons, we shall consider $L(2,1)$-labelings with labels starting with one. To avoid confusion, we shall call such labelings $L(2,1)$-*colorings*. Formally, a k-$L(2,1)$-coloring of a graph G is any function $c : V \to [k]$ such that

1. $|c(v) - c(w)| \geq 1$ for all $v, w \in V(G)$ such that $d(u, w) = 2$,
2. $|c(v) - c(w)| \geq 2$ for all $v, w \in V(G)$ such that $vw \in E(G)$.

Definition 2 *A b-fold coloring of the plane φ is called* a b-fold $L^*(2,1)$-coloring of the plane with color set $[k]$ *if for any two points $p_1, p_2 \in \mathbb{R}^2$:*

- $\mathrm{dist}(p_1, p_2) = 1 \Rightarrow \forall_{i_1,i_2\in\{1,\ldots,b\}}\ 2 \leq |\varphi_{i_1}(p_1) - \varphi_{i_2}(p_2)| < k - 1$,
- $1 < \mathrm{dist}(p_1, p_2) \leq 2 \Rightarrow \forall_{i_1,i_2\in\{1,\ldots,b\}}\ 1 \leq |\varphi_{i_1}(p_1) - \varphi_{i_2}(p_2)|$.

By $L^*(2,1)$-coloring of the plane we mean 1-fold $L^*(2,1)$-coloring of the plane.

3 On-line coloring

The main idea of the algorithm is as follows. We start with some fixed tiling-based b-fold coloring $\varphi = (\varphi_1, \ldots, \varphi_b)$ of the plane with colors $[k_\varphi]$. When a disc D is read, it is assigned to one of the b layers of φ (we try to distribute discs to layers as uniformly as possible). Then a tile from this layer that contains a center of D is found. The vertex corresponding to D is colored with the color of this tile plus k_φ multiplied by the number of vertices previously assigned to this tile.

Algorithm $Color_\varphi((D_i)_{i\in[n]})$
1. ForEach $i \in [n]$
2. Read D_i, let v_i be the center of D_i
3. ForEach $r \in [b]$ let $T_r(v_i)$ be the tile from the layer r containing v_i
4. $\ell(v_i) \leftarrow 1 + (|\{v_1, \ldots v_{i-1}\} \cap \bigcap_{r\in[b]} T_r(v_i)| \pmod b)$
5. $t(v_i) \leftarrow |\{u \in \{v_1, \ldots v_{i-1}\} \cap T_{\ell(v_i)}(v_i) : \ell(u) = \ell(v_i)\}|$
6. $c(v_i) \leftarrow \varphi_{\ell(v_i)}(v_i) + k_\varphi \cdot t(v_i)$
7. Return c

Theorem 1 *Let φ be a tiling-based b-fold coloring of the plane with color set $[k_\varphi]$, and $(D_i)_{i\in[n]}$ be a sequence of unit discs. Algorithm $Color_\varphi((D_i)_{i\in[n]})$ returns a coloring of $G := G((D_i)_{i\in[n]})$. Moreover, if φ is a b-fold $L^*(2,1)$-coloring of the plane, then Algorithm $Color_\varphi((D_i)_{i\in[n]})$ returns an $L(2,1)$-coloring of G.*

Theorem 2 *Let φ be a b-fold coloring of the plane with color set $[k_\varphi]$, with at most γ_φ subtiles in one*

tile, and let $(D_i)_{i\in[n]}$ be a sequence of unit discs. Algorithm $Color_\varphi(\ (D_i)_{i\in[n]})$ returns coloring of $G := G((D_i)_{i\in[n]})$ with the highest color not exceeding $k_\varphi \cdot \lfloor \frac{\omega(G)+(b-1)\gamma_\varphi}{b} \rfloor$.

Proof. Let $\gamma = \gamma_\varphi$ and let v_i be a vertex that got the biggest color. Consider the moment of the course of the algorithm when vertex v_i was colored. Let $\ell(v_i), t(v_i), c(v_i)$ be defined as in the algorithm. Let $T = T_{\ell(v_i)}$ be the tile from the $\ell(v_i)$-th layer containing v_i. Let $S_1, \dots S_\gamma$ be the subtiles of T. Let $s_q = |\{u \in \{v_1, \dots v_i\} : u \in S_q\}|$ and $s_q(\ell(v_i)) = |\{u \in \{v_1, \dots v_i\} : u \in S_q, \ell(u) = \ell(v_i)\}|$ for $q \in [\gamma]$.

Notice that, thanks to the formula in line 4 of the algorithm, vertices in the subtile $\bigcap_{r\in[b]} T_r(v_i)$ are almost uniformly distributed among layers.

The key observation is that by the definition of $\ell(v_i)$ we get $s_q \geq b\cdot(s_q(\ell(v_i))-1)+\ell(v_i)$. Now we are ready to estimate the number of vertices from $\{v_1, \dots, v_i\}$ contained in $T_{\ell(v_i)}$. Notice that these vertices are pairwise at distance less than one and hence they form a clique. We obtain

$$\begin{aligned}\omega(G) \geq& \sum_{q=1}^{\gamma} s_q \geq \sum_{q=1}^{\gamma} b\cdot(s_q(\ell(v_i))-1)+\ell(v_i)\\ \geq& b\cdot\left[\sum_{q=1}^{\gamma}(s_q(\ell(v_i))-1)\right]+\gamma\\ =& b\cdot(t(v_i)+1)-(b-1)\gamma\end{aligned}$$

and thus

$$t(v_i) \leq \left\lfloor \frac{\omega(G)+(b-1)\gamma}{b} - 1 \right\rfloor.$$

Finally we obtain $c(v_i) \leq k_\varphi \cdot \lfloor \frac{\omega(G)+(b-1)\gamma}{b} \rfloor$ which, by the choice of v_i, is the highest color used. □

This shows that is it crucial to construct good b-fold colorings of the plane. We are able to do this if b is a square number.

Theorem 3 *For $h \in \mathbb{N}_+$ there exists a tiling-based h^2-fold coloring of the plane with $\left\lceil (\frac{2}{\sqrt{3}}+1)\cdot h \right\rceil^2$ colors and $\gamma_\varphi = 6h^2$.*

Directly from Theorems 2 and 3 we obtain:

Corollary 4 *For the h^2-fold φ coloring of the plane from Theorem 3 we have*

$$\begin{aligned}&\mathrm{cr}(Color_\varphi(\mathcal{G}_\omega)) \leq\\ &\frac{\left\lceil (\frac{2}{\sqrt{3}}+1)\cdot h \right\rceil^2}{\omega} \cdot \left\lfloor \frac{\omega+(h^2-1)6h^2}{h^2} \right\rfloor\\ &= 4.65 + O\left(\frac{1}{h}\right) + O\left(\frac{h^4}{\omega}\right).\end{aligned}$$

Notice that for $h = 5$ and graphs G with $\omega(G) \geq 108901$, the competitive ratio of the algorithm is less than 5.

Analogously to Theorem 3, we are able to construct a good b-fold $L^*(2,1)$-coloring of the plane.

Theorem 5 *There exists b-fold tiling-based $L^*(2,1)$-coloring φ of the plane for*

1. *$b = 1$ with color set [20] (see Figure 1),*
2. *$b = 2$ with color set [34] and the parameter $\gamma_\varphi = 4$ (see Figure 2),*
3. *$b = 3$ with color set [49] and the parameter $\gamma_\varphi = 6$,*
4. *$b = h^2$ for $h \in \mathbb{N}$ with $3\rho^2+1$ colors, where $\rho = \left\lceil h(\frac{2}{\sqrt{3}}+1)+1 \right\rceil$, and $\gamma_\varphi = 6h^2$*

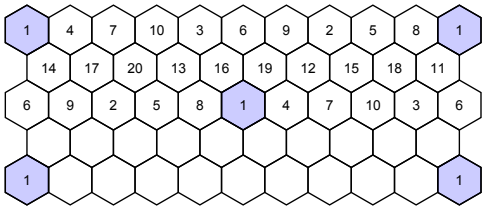


Figure 1: 1-fold $L^*(2,1)$-coloring of the plane

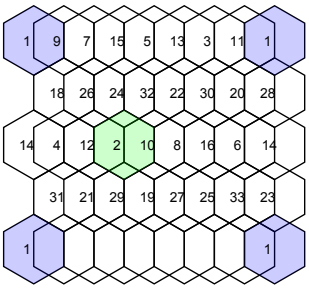


Figure 2: 2-fold $L^*(2,1)$-coloring of the plane

Corollary 6 *For $b \in \mathbb{N}$ and b-fold $L^*(2,1)$-colorings φ of the plane from Theorem 5, the value $\mathrm{cr}(Color_\varphi(\mathcal{G}_\omega))$ is at most:*

1. *$10 + \frac{10}{2\omega-1}$ for φ from Theorem 5.1,*
2. *$8.5 + \frac{76.5}{2\omega-1}$ for φ from Theorem 5.2,*
3. *$8\frac{1}{6} + \frac{204.17}{2\omega-1}$ for φ from Theorem 5.3,*
4. *$(3\lceil h(\frac{2}{\sqrt{3}}+1)+1\rceil^2+1)\frac{\omega+6h^2(h^2-1)}{h^2(2\omega-1)}$ $= 6.97 + O\left(\frac{1}{h}\right) + O\left(\frac{h^4}{\omega}\right)$ for φ from Theorem 5.4.*

4 Off-line $L(2,1)$-labeling

In this section we give an improvement for the following theorem by Shao *et al.* [14], which partially answers the question of Calamoneri [1, Section 4.7.1].

Theorem 7 (Shao *et al.* [14]) *If $G \in UDG$, then $\lambda(G) \leq \frac{4}{5}\Delta^2 + 2\Delta$.*

By Δ we denote the maximum degree of the input graph G. Fix some vertex v. By V_L we denote the half-plane lying left of v (including the boundary). A neighbor w of v is a *left neighbor* if $w \in V_L$. A neighbor w of v is *important*, if it is a left neighbor of v, or w has a neighbor $w' \in V_L$, such that $\text{dist}(w', u) > 1$ for every left neighbor u of v (in particular, w' is not a neighbor of v). It is easy to verify that each vertex v has at most 3 pairwise non-adjacent left neighbors. The following lemma is the strengthening of this observation.

Lemma 8 *Let $G \in UDG$. Each vertex has at most 4 pairwise non-adjacent important neighbors.*

Now we can present the first bound.

Lemma 9 *Let $G \in UDG$ and $\Delta \geq 7$. Then $\lambda(G) \leq \frac{3}{4}\Delta^2 + 3(\Delta - 1)$.*

Proof. We use a greedy algorithm, processing vertices from left to right. Consider a vertex v. By N_1 we denote the set of the left neighbors of v, and by N_2 we denote the set of important right neighbors of v and by N^2 we denote the set of vertices left of v, which are not in N_1, but share a common neighbor with v. Observe that our algorithm will never use a label bigger than $3|N_1|+|N^2|$. Let $d := |N_1 \cup N_2| \leq \Delta$.

Suppose that $N_2 \neq \emptyset$. Let $H = G[N_1 \cup N_2]$. By Lemma 8, it does not contain an independent set of size 5, so its complement, $\bar{H}$, is K_5-free. By the famous theorem of Turán, the maximum number of edges in $\bar{H}$ is $\frac{3}{4}\frac{d^2}{2}$. So the number of edges in H is at least $\binom{d}{2} - \frac{3}{4}\frac{d^2}{2} = \frac{d}{2}\left(\frac{d}{4} - 1\right)$. This gives us the following: $|N^2| \leq d(\Delta - 1) - 2 \cdot \frac{d}{2}\left(\frac{d}{4} - 1\right) \leq \frac{3}{4}\Delta^2$. Since $|N_2| > 0$ and thus $|N_1| \leq \Delta - 1$, we obtain that $3|N_1| + |N^2| \leq 3(\Delta - 1) + \frac{3}{4}\Delta^2 = \frac{3}{4}\Delta^2 + 3\Delta - 3$.

It is easy to verify that if $N_2 = \emptyset$, then $3|N_1| + |N^2| \leq \frac{2}{3}\Delta^2 + 3\Delta < \frac{3}{4}\Delta^2 + 3\Delta - 3$ for $\Delta \geq 7$. □

Fiala *et al.* [3] proved that if $G \in UDG$, then $\lambda(G) \leq 18\omega(G)$. Since $\omega(G) \leq \Delta + 1$, we obtain the following corollary.

Corollary 10 *Let G be a unit disk graph with maximum degree at most Δ. Then $\lambda(G) \leq 18\Delta + 18$.*

Combining the bound $\lambda(G) \leq \Delta^2 + 2\Delta - 2$ by Gonçalves [4], the bound from Lemma 9 and the bound from Corollary 10, we get the following.

Theorem 11 *If $G \in UDG$, then $\lambda(G) \leq f(\Delta)$ for*

$$f(\Delta) = \begin{cases} \Delta^2 + 2\Delta - 2 & \text{if } \Delta < 7, \\ \frac{3}{4}\Delta^2 + 3\Delta - 3 & \text{if } 7 \leq \Delta < 22, \\ 18\Delta + 18 & \text{if } \Delta \geq 22. \end{cases}$$

References

[1] T. Calamoneri, The $L(h,k)$-Labelling Problem: An Updated Survey and Annotated Bibliography, *Comput. J.*, 54: 1344–1371, 2011.

[2] G. Exoo, ε-Unit Distance Graphs, *Discrete Comput. Geom.*, 33: 117–123, 2005.

[3] J. Fiala, A. Fishkin, F. Fomin, On distance constrained labeling of disk graphs, *Theoretical Computer Science* 326: 261–292, 2004.

[4] D. Gonçalves, On the $L(p,1)$-labelling of graphs. *Discrete Mathematics* 308: 1405–1414, 2008.

[5] R. Grak, M. Luckner, Malfunction Immune Wi-Fi Localization Method, *Computational Collective Intelligence*, LNCS 9329, 328-337, 2015

[6] J.R. Griggs, R.K. Yeh, Labeling graphs with a condition at distance two, *SIAM J. Discrete Math.*, 5: 586–595, 1992.

[7] J. Grytczuk, K. Junosza-Szaniawski, J. Sok, K. Wsek, *Fractional and j-fold coloring of the plane*, to appear in *DCG*.

[8] H. Hadwiger, Ungeloste Probleme, *Elemente der Mathematik*, 16: 103–104, 1961.

[9] W.K. Hale, Frequency assignment: Theory and applications, *Proc. IEEE*, 68: 1497–1514, 1980.

[10] T.A. McKee, F.R. McMorris, Topics in Intersection Graph Theory. *SIAM Monographs on Discrete Mathematics and Applications, vol. 2*, SIAM, Philadelphia, 1999.

[11] E. Malesińska, Graph theoretical models for frequency assignment problems, *Ph.D. Thesis*, Technical University of Berlin, Germany, 1997.

[12] R. Peeters, On coloring j-unit sphere graphs, *Technical Report*, Department of Economics, Tilburg University, 1991.

[13] F.S. Roberts, T-colorings of graphs: Recent results and open problems, *Disc. Math.*, 93: 229–245, 1991.

[14] Z. Shao, R. Yeh, K.K. Poon, W.C. Shiu, The $L(2,1)$-labeling of $K_{1,n}$-free graphs and its applications. *Applied Math. Letters*, 21: 1188–1193, 2008.

Approximating Smallest Containers for Packing Three-dimensional Convex Objects

Helmut Alt* Nadja Scharf*

Abstract

We investigate the problem of computing a minimum volume container for the non-overlapping packing of a given set of three-dimensional convex objects. Already the simplest versions of the problem are $\mathcal{NP}$-hard so that we cannot expect to find exact polynomial time algorithms. We give constant ratio approximation algorithms for packing axis-parallel (rectangular) cuboids under translation into an axis-parallel (rectangular) cuboid as container, for cuboids under rigid motions into an axis-parallel cuboid or into an arbitrary convex container, and for packing convex polyhedra under rigid motions into an axis-parallel cuboid or arbitrary convex container. This work gives the first approximability results for the computation of minimum volume containers for the objects described.

1 Introduction

The problem of efficiently packing objects arises in a large variety of contexts. Apart from the obvious ones, like transportation or storage, there are more abstract ones, for example cutting stock or scheduling. Consequently, packing problems have been investigated in mathematics and operations research for a long time (for a survey and references, see [1]).

In this work, we consider the problem of packing three-dimensional convex polyhedra into a minimum-volume container. All variants of this problem are NP-hard. We will develop constant factor approximation algorithms for some of them. The worst case constant factors are still very high, but probably they will be much lower for realistic inputs. The major aim of this paper, however, is to show the existence of constant factors at all, i.e., that the problems belong to the complexity class APX. For a complete version, see [3].

Related Work. So far, there are only few results about finding containers of minimum volume. Related problems include strip packing and bin packing. In two-dimensional strip packing the width of a strip is given and the objects should be packed in order to minimize the length of the strip used. In three dimensions, the rectangular cross section of the strip is fixed. Bin-packing is the problem where the complete container is fixed and the objective is to minimize the number of containers to pack all objects.

Approximation algorithms have been developed for two- and three-dimensional bin and strip packing (e.g. [4, 5, 6, 7]). Approximation algorithms for minimum area containers in two dimensions were given by v.Niederhäusern [8] and Alt et al. [2].

The well-known $\mathcal{NP}$-complete problem PARTITION can be reduced to our problem showing $\mathcal{NP}$-hardness.

2 Preliminaries and Notation

For most algorithms considered here, the input is a set of rectangular boxes $\mathcal{B} = \{b_1, b_2, \dots b_n\}$. We denote a box b_i in axis-parallel orientation by its height, width and depth (h_i, w_i, d_i).

We define:
$h_{\max} = \max\{h_i \mid b_i \in \mathcal{B}\}$,
$w_{\max} = \max\{w_i \mid b_i \in \mathcal{B}\}$, and
$d_{\max} = \max\{d_i \mid b_i \in \mathcal{B}\}$.

Definition 1 (OMCOP) *An instance of orthogonal minimal container packing (*OMCOP*) is a set of convex polyhedra. The aim is to pack these polyhedra non-overlapping such that the minimal axis-parallel container has minimal volume V_{opt}. Variants include the kind of motions allowed or that more specialized objects are to be packed.*

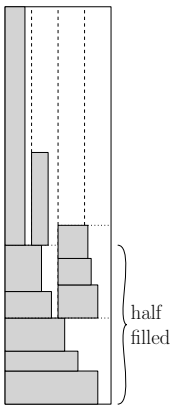


Figure 1: Result of Alg. 1

Algorithm 1 was first given in [8]. We describe it here in detail since it will be used later as a subroutine. For an example see Figure 1.

Observation 1 *The resulting strip of Algorithm 1 is half filled with rectangles up to the bottom edge of the highest rectangle r_i touching the upper end of the packing. Otherwise, r_i could have been placed lower. Thus, the strip is half filled except for a part with area at most $w \cdot h_{\max}$.*

*Institute of Computer Science, Freie Universität Berlin, alt@mi.fu-berlin.de, nadja.scharf@fu-berlin.de. This research was partially funded by DFG (Deutsche Forschungsgemeinschaft) under grant no. AL 253/7-2.

Algorithm 1:

Input: List $\mathcal{S}$ of rectangles $r_i = (w_i, h_i)$, a width for the strip w

1. Split $\mathcal{S}$ in sublists $\mathcal{S}_j = \left\{r_i \in \mathcal{S} \mid \frac{w}{2^{j-1}} \geq w_i > \frac{w}{2^j}\right\}$ for $j \geq 1$.
2. Start with packing the rectangles in $\mathcal{S}_1$ on top of each other in the strip $[0, w] \times [0, \infty)$. Split the remaining strip in two substrips with width $\frac{w}{2}$ and pack the rectangles in $\mathcal{S}_2$ one after another into these substrips. Each r_i is packed in the substrip with current minimal height.
3. Again split the substrips into two and pack S_3. Iterate that process until everything is packed.

3 Reduction from 3D-OMCOP to Strip Packing

In this section we consider the version of OMCOP where the given objects are axis-parallel boxes that are to be packed under translation. The idea of the reduction of OMCOP to strip packing is to test different base areas for the strip and to return the result with minimal volume. The base area of an optimal solution is a rectangle of width within the interval $\left[w_{\max}, W_{\sum}\right]$ and depth within the interval $\left[d_{\max}, D_{\sum}\right]$, where $W_{\sum}$ ($D_{\sum}$) denotes the sum of width (depth) of all boxes to be packed. We subdivide these intervals logarithmically depending on some parameter ε and call for all resulting width-depth-pairs as base area a strip packing algorithm with the given boxes. For a possible subdivision see Figure 2. With a more detailed elaboration and analysis (see [3]) we obtain the following theorem.

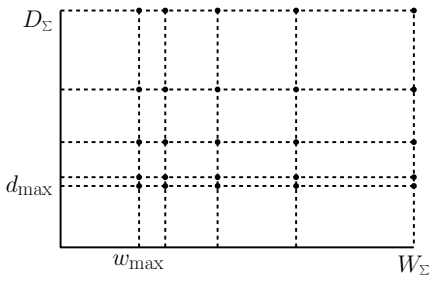


Figure 2: Example for a subdivision. The tested base areas have their lower left corner in common, candidates for the upper right corner are the grid points.

Theorem 1 *If we use an α-approximation algorithm to pack the boxes under translation into the strips with the base areas defined above, we obtain for any fixed $\varepsilon > 0$ an $(\alpha + \varepsilon)$-approximation for the* OMCOP *variant where n axis aligned boxes are to be packed under translation. Its runtime is $\mathcal{O}\left(T(n)\frac{\log^2 n}{\varepsilon^2}\right)$ where $T(n)$ is the runtime of the strip packing algorithm.*

If we use the algorithm given by Diedrich et al. [5] which gives a $\frac{29}{4}$-approximation for three-dimensional strip packing, we obtain the following corollary.

Corollary 2 *There exists a $(7.25 + \varepsilon)$-approximation algorithm for packing axis-parallel boxes under translation into a minimum volume axis-parallel box with runtime polynomial in the input size and $\frac{1}{\varepsilon}$.*

4 Algorithms for Variants of OMCOP

In this section we will give algorithms for variants of OMCOP. The basic idea is to get rid of the third dimension by partitioning the set of objects into sets of objects with similar height and then packing those using an algorithm for two-dimensional boxes. These containers then get cut into pieces with equal base area. The pieces will be stacked on top of each other, see Algorithm 2.

4.1 Cuboids under Translation

Although this algorithm gets outperformed by the construction in the previous section, we state it here as base for the algorithms for the other variants. For an illustration of steps 3 to 5 see Figure 3.

Algorithm 2:

Input: Set of axis parallel boxes $\mathcal{B} = \{b_1, \ldots, b_n\}$, $\alpha \in (0, 1)$, $c > 1$

1. Partition $\mathcal{B}$ into subsets of boxes that have almost the same height: $\mathcal{B}_j = \left\{b_i \in \mathcal{B} \mid h_{\max} \cdot \alpha^j < h_i \leq h_{\max} \cdot \alpha^{j-1}\right\}$
2. Use Algorithm 1 to pack every $\mathcal{B}_j$ into a strip with width $w_{\max}$ and height $h_{\max} \cdot \alpha^{j-1}$ by taking the base areas of the boxes as rectangles and applying Algorithm 1 to them.
3. Divide the strips into pieces with depth $(c-1) \cdot d_{\max}$, ignoring the last part of the strip of depth $d_{\max}$. (Parts of boxes contained in this part of the strip will be covered in step 4.)
4. Extend each piece to depth $c \cdot d_{\max}$ such that every box lies entirely in the piece its front lies in.
5. Stack the pieces on top of each other.

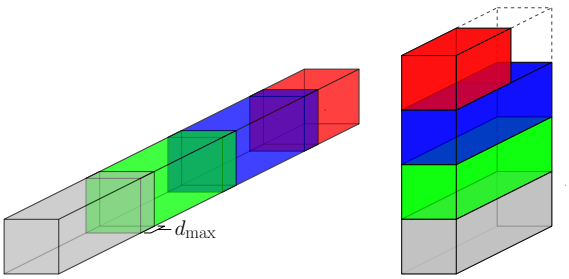


Figure 3: Cut strip and obtained pieces stacked.

Theorem 3 *For suitable values of c and α Algorithm 2 computes a 11.542-approximation for the variant of* OMCOP *where n axis parallel cuboids are packed under translation in $\mathcal{O}(n \log n)$ time.*

Proof. Let D_j denote the depth of the strip obtained in step 2 for the boxes in $\mathcal{B}_j$. Then we get by step 3 $k = \left\lceil \frac{D_j - d_{\max}}{(c-1)d_{\max}} \right\rceil$ pieces. After step 4 each piece has volume $c \cdot d_{\max} w_{\max} h_{\max} (\alpha)^{j-1}$. Then the total volume of the pieces obtained for the subset $\mathcal{B}_j$ is:

$$\begin{aligned} V_j =& k \cdot c \cdot d_{\max} w_{\max} h_{\max} (\alpha)^{j-1} \\ <& \frac{c}{c-1} (D_j - d_{\max}) w_{\max} h_{\max} (\alpha)^{j-1} \\ &+ c \cdot d_{\max} w_{\max} h_{\max} (\alpha)^{j-1}. \end{aligned}$$

We know from Algorithm 1 that the base area of the strip is half filled with boxes except for the last part of depth $d_{\max}$ (Observation 1), so $(D_j - d_{\max}) w_{\max} \leq 2 \sum_{b_i \in \mathcal{B}_j} A_B(b_i)$ where $A_B(b)$ denotes the base area of box b. Also, for every $b_i \in \mathcal{B}_j$ the inequality $h_{\max} (\alpha)^{j-1} < \frac{h_i}{\alpha}$ holds. Thus, we get for the total volume of the packing V that

$$\begin{aligned} V \leq& \sum_{j=1}^{\infty} \Bigg(\frac{c}{c-1} (D_j - d_{\max}) w_{\max} h_{\max} \alpha^{j-1} \\ &+ c \cdot d_{\max} w_{\max} h_{\max} \cdot \alpha^{j-1} \Bigg) \\ \leq& \sum_{j=1}^{\infty} \Bigg(\frac{2c}{\alpha (c-1)} \sum_{b_i \in \mathcal{B}_j} V(b_i) \\ &+ c \cdot w_{\max} d_{\max} h_{\max} \cdot \alpha^{j-1} \Bigg) \\ \leq& \frac{2c}{\alpha (c-1)} \underbrace{\sum_{b \in \mathcal{B}} V(b)}_{\leq V_{\text{opt}}} \\ &+ c \cdot \underbrace{w_{\max} \cdot d_{\max} \cdot h_{\max}}_{\leq V_{\text{opt}}} \cdot \sum_{l=0}^{\infty} \alpha^l \qquad (1) \\ \leq& \left(\frac{2c}{\alpha (c-1)} + \frac{c}{1-\alpha} \right) V_{\text{opt}}. \qquad (2) \end{aligned}$$

The factor before V_{opt} in term (2) is minimized if the partial derivatives with respect to c and α are 0. This gives an approximation ratio of $\frac{3}{\sqrt[3]{2}-1} \approx 11.542$. □

4.2 Cuboids under Rigid Motions

Now we consider the variant of OMCOP where the objects to be packed are boxes and rigid motions are allowed. We use the algorithm stated above but with an extra preprocessing step, namely rotating every box $b_i \in \mathcal{B}$ such that it becomes axis parallel and $h_i \geq w_i \geq d_i$. This can be done in $\mathcal{O}(n)$ time. To prove the performance bound we need Lemma 4.

Lemma 4 *If every $b_i \in \mathcal{B}$ is oriented such that $h_i \geq w_i \geq d_i$, then $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq \sqrt{6} \cdot V_{opt}$.*

Proof. An optimal container has to contain the box determining $h_{\max}$, so it contains a line segment l_1 of length $h_{\max}$. The projection of l_1 on one of the axes has a length of at least $\frac{1}{\sqrt{3}} h_{\max}$. W.l.o.g. let this axis be the x-axis. Thus, the optimal container has an expansion of at least $\frac{1}{\sqrt{3}} h_{\max}$ in x-direction. Since every box is higher then wide, a box with width $w_{\max}$ contains a disk D with diameter $w_{\max}$ and so the optimal container does. D contains a diametric line segment l_2 which is parallel to the y-z-plane. The projection of l_2 and thus the one of the whole box on the y-axis or on the z-axis has a length of at least $\frac{1}{\sqrt{2}} w_{\max}$. W.l.o.g. let this be the y-axis. A box with depth $d_{\max}$ contains a sphere s with diameter $d_{\max}$. The projection of s on any axis, in particular the z-axis, has length at least $d_{\max}$. □

Observe that every argument leading to inequality (1) still holds for this variant of the algorithm. Using Lemma 4 to estimate $h_{\max} \cdot w_{\max} \cdot d_{\max}$ we get an approximation factor of $\frac{2c}{\alpha(c-1)} + \frac{c \cdot \sqrt{6}}{1-\alpha}$. Minimizing this expression as before yields:

Theorem 5 *The given algorithm computes a 17.738-approximation for the variant of 3D* OMCOP *where n axis parallel cuboids are packed under rigid motions in $\mathcal{O}(n \log n)$ time.*

Convex Container. If we allow a convex container instead of an orthogonal container, we can use the same algorithm but adapt the analysis. The arguments leading to inequality (1) still hold since they only use the total volume of the boxes as estimate for the volume of an optimal container. But we can only show $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq 6 \cdot V_{\text{opt}}$, so we get with a detailed analysis the following theorem.

Theorem 6 *Using the algorithm described in section 4.2 we get a 29.135-approximation for packing n axis parallel boxes under rigid motions into a smallest-volume convex container in time $\mathcal{O}(n \log n)$.*

4.3 Convex Polyhedra under Rigid Motions

We use the algorithm from section 4.2 to pack convex polyhedra under rigid motions into an axis-parallel minimal volume box. To do so, we add another preprocessing step where we compute an enclosing box for every polyhedron. We then pack these boxes with the algorithm discussed in section 4.2. For a convex polyhedron p the enclosing box is built as follows: Let B and T be two points of p with largest distance h and π a hyperplane normal to the line segment $\overline{BT}$. Let p' be the orthogonal projection of p onto π, R' and L'

be two points of p' with largest distance w, and R and L its preimages. Let l be a line normal to $\overline{R'L'}$, the projection of p' onto l a line segment of length d with endpoints F'' and D'', and F and D its perimages in p. Then the enclosing box is $B_p = (h, w, d)$. See Figure 4 for an example. Checking all pairs of vertices as candidates for B and T, and R' and L', we get a total running time of $\mathcal{O}\left(m^2\right)$ for computing the bounding boxes of polyhedra with m vertices in total. For the analysis of this algorithm we need two lemmata that follow.

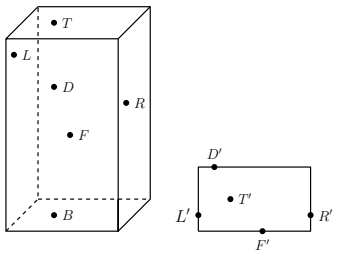


Figure 4: Box with a point of the enclosed polyhedron in every facet and the projection of the box on its base.

Lemma 7 *Let $b = (h, w, d)$ with $h \geq w \geq d$ be the enclosing box obtained for polyhedron p. Then, parallel to any given plane, p contains a line segment of length at least $w \cdot \frac{1}{\sqrt{5}}$.*

This lemma can be proven by showing that either each height in triangle (TBL) or triangle (TBR) is at least $\frac{w}{\sqrt{5}}$. The complete proof can be found in [3].

Lemma 8 *Let $b = (h, w, d)$ be the enclosing box obtained for a polyhedron p. The projection of p onto an arbitrary line g has length at least $\frac{d}{8\sqrt{3}}$.*

This Lemma is shown by an elaborate construction, where we find four line segments inside p such that the projection of at least one of them onto g has length at least $\frac{1}{8\sqrt{3}}d$. The complete proof can be found in [3].

Just as in the proof of Lemma 4 any container, in particular the optimal one, must contain a line segment of length $h_{\max}$ whose projection on one axis, say the x-axis, has length at least $\frac{h_{\max}}{\sqrt{3}}$. Applying Lemma 7 to the y-z-plane and the polyhedron defining $w_{\max}$ gives a line segment of length at least $\frac{w_{\max}}{\sqrt{5}}$ whose projection onto at least one axis, say the y-axis, has length at least $\frac{w_{\max}}{\sqrt{10}}$. By Lemma 8, the projection of the polyhedron defining $d_{\max}$ onto the z-axis has length at least $\frac{d_{\max}}{8\sqrt{3}}$. Summarizing, we obtain that $V_{\text{opt}} \geq \frac{1}{24\sqrt{10}}\, h_{\max} \cdot w_{\max} \cdot d_{\max}$. Using this inequality and the fact that the volume of each enclosing box is at most 6 times the volume of the enclosed polyhedron, we derive the following approximation ratio from inequality (1): $\frac{12c}{\alpha(c-1)} + \frac{c \cdot 24\sqrt{10}}{1-\alpha}$. We get by minimization:

Theorem 9 *The given algorithm computes a 277.59-approximation for the variant of* OMCOP *where n convex polyhedra having m vertices in total are to be packed under rigid motions in time $\mathcal{O}\left(m^2 + n \log n\right)$.*

Convex Container. We use the the result of the algorithm given in Section 4.3 to compute an approximation for a minimum volume arbitrary convex container. The approximation ratio becomes a different expression since we can only show $h_{\max} \cdot w_{\max} \cdot d_{\max} \leq 24\sqrt{60} V_{\text{opt}}$. A detailed analysis yields the following theorem.

Theorem 10 *The algorithm given in section 4.3 computes a convex container with volume at most 511.37 times the volume of an optimal convex container for packing n convex polyhedra having m vertices in total under rigid motions in time $\mathcal{O}\left(m^2 + n \log n\right)$.*

References

[1] H. Alt. Computational aspects of packing problems. *Bulletin of the EATCS*, 118, 2016. forthcoming.

[2] H. Alt, M. de Berg, and C. Knauer. Approximating minimum-area rectangular and convex containers for packing convex polygons. In *Proc. 23th Annu. European Sympos. Algorithms (ESA)*, pages 25–34, 2015.

[3] H. Alt and N. Scharf. Approximating Smallest Containers for Packing Three-dimensional Convex Objects. *ArXiv e-prints*, January 2016.

[4] N. Bansal and A. Khan. Improved approximation algorithm for two-dimensional bin packing. In *Proc. 25th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 13–25, 2014.

[5] F. Diedrich, R. Harren, K. Jansen, R. Thöle, and H. Thomas. Approximation algorithms for 3D orthogonal knapsack. *J. Comput. Sci. Tech.*, 23(5):749–762, 2008.

[6] K. Jansen and L. Prädel. A new asymptotic approximation algorithm for 3-dimensional strip packing. In *Proc. 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 327–338, 2014.

[7] F. K. Miyazawa and Y. Wakabayashi. Three-dimensional packings with rotations. *Comput. Oper. Res.*, 36(10):2801–2815, 2009.

[8] L. von Niederhäusern. Packing polygons: Research of approximation algorithms. Master's thesis, Freie Universität Berlin and École Polytechnique Fédérale de Lausanne, 2014.

Packing Plane Spanning Double Stars into Complete Geometric Graphs

Patrick Schnider*

Abstract

Consider the following problem: Given a complete geometric graph, how many plane spanning trees can be packed into its edge set?

We investigate this question for *plane spanning double stars* instead of general spanning trees. We give a necessary, as well as a sufficient condition for the existence of packings with a given number of plane spanning double stars. We also construct complete geometric graphs with an even number of vertices that cannot be partitioned into plane spanning double stars.

1 Introduction

A *geometric graph* is a drawing of a graph in $\mathbb{R}^2$ where the vertex set is drawn as a point set in general position, that is, no three points lie on a line, and each edge is drawn as a straight-line segment. A geometric graph is called *plane* if no pair of edges crosses. For two vertices v and w in a geometric graph G, we say that v *sees* w *in* G if the line segment between v and w is not crossed by any edge of G. In this paper we will assume all point sets to be in general position, and for a point set $\mathcal{P}$, we denote by $K(\mathcal{P})$ the complete geometric graph with vertex set $\mathcal{P}$.

It is a long-standing open question whether any complete geometric graph with an even number of vertices has a partition of its edge set into plane spanning trees. If the vertices lie in convex position, the question can be answered in the affirmative, and all possible partitions can be characterized, as was done by Bose et al. [3]. The authors also give a sufficient condition for a complete geometric graph to have a partition of its edge set into plane spanning trees:

Theorem 1 ([3]) *Let $\mathcal{P}$ be a point set with $n = 2m$ points. Suppose that there is a set $\mathcal{L}$ of pairwise non-parallel lines with exactly one point of $\mathcal{P}$ in each open unbounded region formed by $\mathcal{L}$. Then $K(\mathcal{P})$ can be partitioned into m plane spanning trees.*

The trees they construct in this case are double stars: A *double star* is either a single edge or a tree such that the induced subgraph of the vertex set without the leaves is a single edge, called the *spine*.

*Department of Computer Science, ETH Zürich, schnpatr@inf.ethz.ch

More generally, one can ask how many plane spanning trees can be packed into the edge set of a complete geometric graph. Aichholzer et al. [1] show that at least $\lfloor\sqrt{\frac{n}{12}}\rfloor$ plane spanning trees can be packed into any complete geometric graph. This result was very recently improved to $\lceil\frac{n}{3}\rceil$ spanning trees by Garcia [4]. Whereas the trees that Garcia uses for his packing have diameter 4, the trees that Aichholzer et al. construct are again double stars. It seems natural to restrict the open question above to the question whether any complete geometric graph can be partitioned into plane spanning double stars. However, the answer to this question is no. We will show this by proving that for any packing with plane spanning double stars, the spines of the double stars form a matching, which we call the *spine matching*. In the case of a partition, this spine matching is a perfect matching. In Section 3 we then show a necessary condition for a matching to be a spine matching and construct a point set which has no perfect matching that satisfies this necessary condition. In Section 4 we show a sufficient condition for a matching to be a spine matching. Finally, in Section 5 we show that we can decide in polynomial time whether a given matching is a spine matching. Due to space restrictions we cannot give all the proofs. We refer the interested reader to the full version [5].

2 Partitions and Packings

Consider a point set $\mathcal{P}$ of size n and a packing of k plane spanning double stars into $K(\mathcal{P})$. Let M be the set of spines of the double stars.

Lemma 2 *The set of spines M of a packing of k plane spanning double stars into $K(\mathcal{P})$ is a matching.*

As mentioned before, we call this matching the *spine matching*.

Proof. We want to show that no two edges of M are incident. Assume for the sake of contradiction that two edges $e = (p, q)$ and $f = (p, r)$ share an endpoint p. Let E and F be the spanning double stars with spines e and f, respectively. Consider the edge $g = (q, r)$. As all double stars in the partition must be spanning, the point r must be connected to the edge e, which means that $f \in E$ or $g \in E$. As f is already the spine of F, we conclude that $g \in E$. On the other hand q must also be connected to the edge

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

f and with the same argument we conclude $g \in F$, which is a contradiction. □

Note that for a partition of $K(\mathcal{P})$ into plane spanning double stars, we need $\frac{n}{2}$ double stars, i.e. the spine matching is a perfect matching. We call a perfect matching on a point set $\mathcal{P}$ *expandable* if it is the spine matching of a partition of $K(\mathcal{P})$ into plane spanning double stars.

Corollary 3 *Let $\mathcal{P}$ be a point set that allows a packing of k plane spanning double stars into $K(\mathcal{P})$. Then there is a subset $\mathcal{P}'$ of $\mathcal{P}$ of size $2k$ that allows a partition of $K(\mathcal{P}')$ into plane spanning double stars.*

Proof. Choose $\mathcal{P}'$ as the set of vertices of the spine matching M. □

On the other hand, we can expand a partition on a subset to a packing on the whole point set.

Lemma 4 *Let $\mathcal{P}$ be a point set and let $\mathcal{P}'$ be a subset of $\mathcal{P}$ of size $2k$ that allows a partition of $K(\mathcal{P}')$ into plane spanning double stars. Then $\mathcal{P}$ allows a packing of k plane spanning double stars into $K(\mathcal{P})$.*

For an illustration of the proof see Figure 1

Proof. Consider an edge e in the spine matching M and a point p in $\mathcal{P}\setminus\mathcal{P}'$. Let E be the plane double star with spine $e = (q, r)$ and let $f = (p, q)$ and $g = (p, r)$ be the edges connecting the point p to the spine e. In order to expand E to a plane spanning double star, we have to add either f or g to E without creating a crossing. Assume for the sake of contradiction that both f and g cross an edge of E. Let s and t be the intersections of f and g with E, respectively. Note that the edge of E that crosses f must be incident to r. Similarly, the edge of E that crosses g is incident to q. As q, r, s and t form a convex quadrilateral, we deduce that E is not plane, which is a contradiction. By induction we can therefore expand E to a plane spanning double star. As the spines form a matching we can do this for every double star in the partition of the subset and the claim follows. □

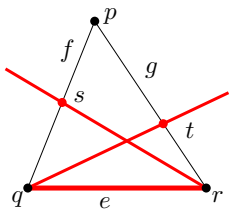


Figure 1: Illustration of the proof of Lemma 4.

Combining Corollary 3 and Lemma 4 we get the following result:

Theorem 5 *Let $\mathcal{P}$ be a point set. Then $K(\mathcal{P})$ allows a packing of k plane spanning double stars if and only if there is a subset $\mathcal{P}'$ of $\mathcal{P}$ of size $2k$ that allows a partition of $K(\mathcal{P}')$ into plane spanning double stars.*

Thus the problem of finding a large packing with plane spanning double stars is equivalent to finding a large subset of the vertex set whose induced graph can be partitioned into plane spanning double stars, i.e. a large expandable matching.

3 A necessary condition

We start by showing that any subset of an expandable matching is again expandable:

Lemma 6 *Let $K(\mathcal{P})$ be partitioned into plane spanning double stars and let $\mathcal{P}'$ be the vertices of any subset of the spine matching M. Then the induced subgraph on $\mathcal{P}'$ inherits a partition into plane spanning double stars.*

Proof. Color each double star in the partition with a different color, including red. Now delete the vertices incident to the red spine and consider the colored subgraph induced by the remaining vertices. This subgraph contains no red edges, as each red edge is incident to the red spine. Also, all deleted edges that are not red cannot be spines. Thus the remaining graph is still partitioned into plane spanning double stars. The result follows by induction. □

Let e be an edge between two points p and q. The *supporting line* ℓ_e of e is the line through p and q.

Let e and f be two edges and let s be the intersection of their supporting lines. If s lies in both e and f, we say that e and f *cross*. If s lies in f but not in e, we say that e *stabs* f and we call the vertex of e that is closer to s the *stabbing vertex* of e. If s lies neither in e nor in f, or even at infinity, we say that e and f are *parallel*. See Figure 2 for an illustration.

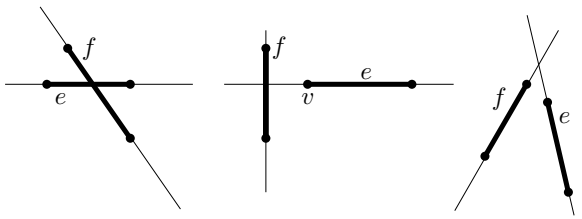


Figure 2: Left: e and f cross; Middle: e stabs f with stabbing vertex v; Right: e and f are parallel.

It can easily be seen that a matching consisting of two parallel edges is not expandable. On the other hand, a matching consisting of two non-parallel edges is expandable, as can be seen in Figure 3.

Lemma 7 *A matching M consisting of two edges e and f is expandable if and only if e and f are not parallel.*

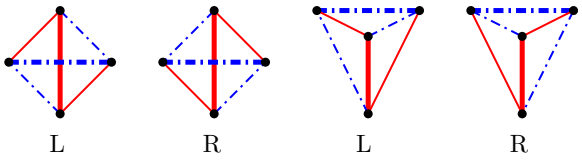


Figure 3: Any pair of crossing or stabbing edges is expandable. The spines are drawn thick.

In Figure 3 we can also see that for two non-parallel edges there are exactly two ways to expand the matching consisting of the two edges into a partition of the induced complete graph. We call them *left-oriented* (L) and *right-oriented* (R).

For larger matchings, the situation is more complicated, but we can still find some configurations that cannot occur in the matching. See Figure 4 for a drawing of these configurations:

A *cross-blocker* is a triple $C = \{e, f, g\}$ of three pairwise non-incident edges such that e and f cross, g stabs both e and f, g does not intersect the convex hull of e and f, and both vertices of g see only one vertex p of e and one vertex q of f in C.

A *stab-blocker* is a triple $S = \{e, f, g\}$ of three pairwise non-incident edges such that f stabs e, g stabs both f and e, g does not intersect the convex hull of e and f, and both vertices of g see only one vertex p of e in S.

Lemma 8 *Let M be a cross-blocker or a stab-blocker. Then M is not expandable.*

For the proof we refer to [5].

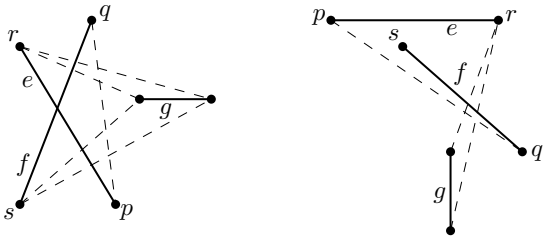


Figure 4: A cross-blocker (left) and a stab-blocker (right).

Combining this with Lemma 7 and Lemma 6, we get a necessary condition for a matching to be expandable.

Theorem 9 *Let $K(\mathcal{P})$ be partitioned into plane spanning double stars. Then the corresponding spine matching M*

- *does not contain two parallel edges,*
- *does not contain a cross-blocker and*
- *does not contain a stab-blocker.*

This allows us to construct a point set whose complete geometric graph cannot be partitioned into plane spanning double stars. For every $k > 0$, we define the *bumpy wheel set* BW_k as follows:

Place $k-1$ points in convex position and partition them into three sets A_1, A_2, A_3 of consecutive points such that $||A_i| - |A_j|| \leq 1$, $i \neq j$. Let H_i, be the convex hull of $\cup_{j \neq i} A_j$. Place the last point p in the interior such that it lies outside of H_i for all $i \in \{1, 2, 3\}$. See Figure 5 for a depiction of BW_{10}.

It can be shown that every parallel-free perfect matching on BW_k, for $k \geq 10$ even, contains a cross-blocker. Thus no perfect matching on these BW_k is expandable. For k odd there cannot even be a perfect matching.

Theorem 10 *For every $k \geq 9$, the complete geometric graph $K(BW_k)$ cannot be partitioned into plane spanning double stars.*

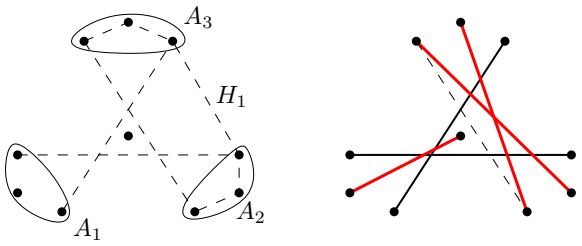


Figure 5: The point set BW_{10} (left) and a cross blocker in a parallel-free matching on this point set (right).

4 A sufficient condition

We will now state a sufficient condition for a matching to be expandable.

A *stabbing chain* are three edges, e, f and g, where e stabs f and f stabs g. We call f the middle edge of the stabbing chain. See Figure 6 for a drawing of some stabbing chains. Note that in the rightmost drawing there are three stabbing chains and each edge is the middle edge in one of the stabbing chains.

Theorem 11 *Let $\mathcal{P}$ be a point set and let M be a perfect matching on $\mathcal{P}$, such that*

(a) *no two edges are parallel,*

(b) *if an edge e stabs two other edges f and g, then the respective stabbing vertices of e lie inside the convex hull of f and g, and*

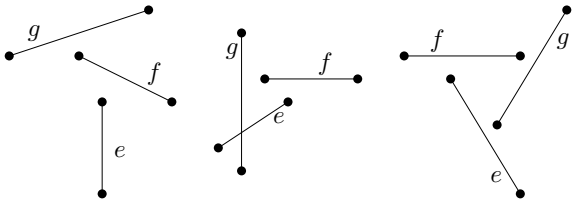


Figure 6: Different stabbing chains.

(c) *if there is a stabbing chain, then the stabbing vertex of the middle edge lies inside the convex hull of the other two edges.*

Then M is expandable.

Note that a stab-blocker is a stabbing chain that satisfies condition (c), but not (b).

We get a partition of $K(\mathcal{P})$ into plane spanning double stars by expanding every pair of edges in M in such a way that the induced K_4 is left-oriented. For a complete proof, we refer to [5]. There it is also shown that the sufficient condition from Bose et al. follows from this result.

Note that this result in particular implies that a set of pairwise crossing edges is expandable. Aronov et al. [2] have shown that every complete geometric graph has a set of at least $\lfloor\sqrt{\frac{n}{12}}\rfloor$ pairwise crossing edges. This proves again the result from Aichholzer et al. [1] that at least $\lfloor\sqrt{\frac{n}{12}}\rfloor$ plane spanning double stars can be packed into any complete geometric graph.

5 Recognizing expandable matchings

In this section we will consider the decision problem where, given a perfect matching on a point set $\mathcal{P}$, we want to decide whether it is expandable. We will show that we can solve this problem in polynomial time.

Recall that there are exactly two ways to expand a pair of non-parallel edges to a partition of their induced K_4 into spanning double stars. We called the two options "left-oriented" and "right-oriented". Expanding a parallel-free perfect matching to a partition into spanning double stars is thus just choosing for each pair of edges in the matching, whether the pair is left-oriented or right-oriented. The given perfect matching is then the spine matching of the partition.

We can check whether a matching of size n is parallel-free by looking at all pairs of edges. As there are $\mathcal{O}(n^2)$ pairs, this can be done in time $\mathcal{O}(n^2)$. If a matching is not parallel-free, it cannot be expandable. So it is enough to only consider parallel-free matchings.

Consider now the partition given by a choice of orientation of each pair of spines in M, where M is parallel-free and has size n, and color each double star with a different color. Assume there is a monochromatic crossing, let us say of color red. Then, as M is parallel-free, the two crossing red edges a and b are incident to exactly three spines: both edges are incident to the red spine e, and each edge is incident to another spine, let us assume that a is incident to the blue spine f, and b is incident to the green spine g. The fact that both a and b are red already determines the orientation of the pairs $\{e, f\}$ and $\{e, g\}$, as a is part of the K_4 induced by e and f and b is part of the K_4 induced by e and g. Also, changing one or both orientations would give a partition where a and b have different colors. Thus each monochromatic crossing can be prevented by forbidding the combination of the orientations of $\{e, f\}$ and $\{e, g\}$ that leads to the crossing being monochromatic. Doing this for every possible monochromatic crossing that could occur in some orientation translates into a 2-CNF with $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ clauses, where every variable corresponds to a pair of edges in the matching. For a 2-CNF we can decide whether it is satisfiable in time linear in the number of clauses, so we can decide in time $\mathcal{O}(n^3)$ whether a parallel-free matching is expandable or not. This proves the following theorem:

Theorem 12 *Given a perfect matching M on a point set $\mathcal{P}$ of size n, it is possible to decide in polynomial time whether this perfect matching can be expanded to a partition of $K(\mathcal{P})$ into plane spanning double stars.*

Acknowledgments

The author would like to thank Emo Welzl and Manuel Wettstein for the fruitful discussions.

References

[1] O. Aichholzer, T. Hackl, M. Korman, M. J. van Kreveld, M. Löffler, A. Pilz, B. Speckmann, and E. Welzl. Packing plane spanning trees and paths in complete geometric graphs. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada, 2014*, 2014.

[2] B. Aronov, P. Erdös, W. Goddard, D. J. Kleitman, M. Klugerman, J. Pach, and L. J. Schulman. Crossing families. In *Proceedings of the Seventh Annual Symposium on Computational Geometry, North Conway, NH, USA, , June 10-12, 1991*, pages 351–356, 1991.

[3] P. Bose, F. Hurtado, E. Rivera-Campo, and D. R. Wood. Partitions of complete geometric graphs into plane trees. *Computational Geometry*, 34(2):116 – 125, 2006.

[4] A. Garcia Olaverri. personal communication.

[5] P. Schnider. Partitions and packings of complete geometric graphs with plane spanning double stars and paths. Master's thesis, ETH Zürich, 2015.

ε-covering is NP-complete

Dominique Attali* Tuong-Bach Nguyen† Isabelle Sivignon‡

Abstract

Consider the dilation and erosion of a shape S by a ball of radius ε. We call ε-covering of S any collection of balls whose union lies between the dilation and erosion of S. We prove that finding an ε-covering of minimum cardinality is NP-complete, using a reduction from vertex cover.

1 Introduction

Unions of balls are common shape representations, useful for instance to describe molecules in biochemistry [3], to quickly detect collisions [1] between shapes or to derive higher-level representations such as medial axis descriptions. The ubiquity of unions of balls is largely due to the existence of provably good conversion algorithms that allow us to derive them from various representations such as point clouds and polygonal meshes [7]. In that case, the union of balls output by the conversion process generally provides only an approximation of the original shape. The quality of the approximation can be measured by its geometric error. In this paper, we introduce a parameter ε that controls the admissible geometric error in a novel way. We say that a collection of balls provides an ε-covering of a shape if its union is contained in the dilation of the shape by b_ε and contains the erosion of the shape by b_ε, where b_ε refers to the ball with radius ε centered at the origin. We are interested in the problem of computing such a collection of balls with minimum cardinality. Our main result is that this problem is NP-complete.

2 Statement of result

In this paper, we suppose that $\mathbb{R}^d$ is endowed with the Euclidean distance. For any point c and real $r > 0$, we denote by $b(c, r)$ the open ball centered at c with radius r. Let $S \subseteq \mathbb{R}^d$ and $\varepsilon > 0$ a real number. The **dilation** of S (by ε) is $S^{\oplus\varepsilon} = \cup_{x\in S} b(x, \varepsilon)$. The **erosion** of S (by ε) is $S^{\ominus\varepsilon} = \{x \mid b(x, \varepsilon) \subseteq S\}$. For any collection of balls $\mathscr{B}$, we adopt the notation $\bigcup \mathscr{B} = \cup_{b\in\mathscr{B}} b$. A collection of balls is rational if each of its balls has a rational radius and a center with rational coordinates. We also assume that these rationals can be represented by integers bounded by a given constant.

Definition 1 *An ε-covering of S is a collection of balls $\mathscr{B}$ such that $S^{\ominus\varepsilon} \subseteq \bigcup \mathscr{B} \subseteq S^{\oplus\varepsilon}$. Additionally if $\mathscr{B}$ is rational, it is a rational ε-covering.*

We are interested in ε-coverings that achieve minimum cardinality and must solve this problem:

Problem 1 (Rational ε-covering problem) *Let $\mathscr{S}$ be a finite rational collection of balls in $\mathbb{R}^d$, $\varepsilon > 0$ a rational and k a positive integer. Does $\bigcup \mathscr{S}$ have a rational ε-covering with at most k balls?*

The purpose of this paper is to prove the following:

Theorem 1 *The rational ε-covering problem in $\mathbb{R}^2$ is NP-complete.*

To prove our theorem, we need to establish that the ε-covering problem is both in NP and NP-hard. For lack of space, we only sketch the proof of the NP property in Section 3 and focus on the NP-hardness in Section 4. Indeed, a formal proof of the former property requires several technical results we will not detail. Note however that the purpose of the rational and $\mathbb{R}^2$ restrictions is to clear these technical hurdles. As for the latter property, it is obtained through a reduction from a variant of the vertex cover problem, which remains valid in all dimensions.

3 NP

The ε-covering problem is in NP if one can verify in polynomial time in the size of $\mathscr{S}$ whether a collection of balls is a solution. Given a shape $S = \bigcup \mathscr{S}$, a parameter $\varepsilon > 0$ and some collection of balls $\mathscr{B}$, is it possible to check the two inclusions $S^{\ominus\varepsilon} \subseteq \bigcup \mathscr{B} \subseteq S^{\oplus\varepsilon}$ in polynomial time? Since we only address the $\mathbb{R}^2$ case in this section, balls are simply disks whose boundaries are circles. We propose a method that relies on the arrangement of those boundary circles.

In general, the exact computation of these arrangements requires an exact handling of real numbers. Though, with our restriction to rationals, we only need to handle algebraic numbers. This can be done using the *isolating interval representation* [8].

An arrangement of circles is the subdivision of $\mathbb{R}^2$ into open connected cells which is induced by these circles. If we consider the original disk supported by

*Gipsa-lab, dominique.attali@gipsa-lab.grenoble-inp.fr
†Gipsa-lab, tuong-bach.nguyen@gipsa-lab.grenoble-inp.fr
‡Gipsa-lab, isabelle.sivignon@gipsa-lab.grenoble-inp.fr

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

each circle, each cell can then be characterized by two sets of disks: the disks that contain the cell, and those that do not. Along with the arrangement itself, this information can be computed in polynomial time for each cell [5, 4]. Owing to this, one can test whether a particular disk b is contained in a finite union of disks $\bigcup \mathscr{S}$. Indeed, it suffices to compute the arrangement of $\mathscr{S} \cup \{b\}$ and then check for every cell covered by b if it is also covered by some disk of $\mathscr{S}$. This can obviously be extended to verifying if a finite union of disks contains another such union. Since $S^{\oplus\varepsilon}$ is the union of the dilated balls of $\mathscr{S}$, we can test if $\mathscr{B}$ satisfies $\bigcup \mathscr{B} \subseteq S^{\oplus\varepsilon}$. As for the second condition, it requires the following technical result.

Proposition 2 *Let $\mathscr{V}$ be the set of vertices of the boundary of S. Consider $\mathscr{S}' = \{b^{\ominus\varepsilon} \mid b \in \mathscr{S}\} \cup \{b(v, \varepsilon) \mid v \in \mathscr{V}\}$. Then there exists a collection $\mathscr{C}$ of cells of the arrangement of $\mathscr{S}'$ such that $S^{\ominus\varepsilon} = \bigcup \mathscr{C}$, and these cells can be found in polynomial time.*

Thus, by computing the arrangement of $\mathscr{S}' \cup \mathscr{B}$ we can test whether $S^{\ominus\varepsilon} \subseteq \bigcup \mathscr{B}$ by inspecting the cells that belong to $S^{\ominus\varepsilon}$. Though a generalization of Proposition 2 to higher dimension seems feasible, the computation of the arrangement of $\mathscr{S}' \cup \mathscr{B}$ is not straightforward.

4 NP-hardness

We prove NP-hardness through a reduction from a variant of the vertex cover problem. Recall that for a graph $G = (V, E)$, a subset $F \subseteq V$ is a vertex cover of G if every edge of E is incident to a vertex of F. Finding a minimum vertex cover is NP-hard, even when restricted to cubic planar graphs [6]. We shall perform the reduction from this particular variant. Let $G = (V, E)$ be a planar graph of degree at most 3. For any $\varepsilon > 0$, we show how to build a finite collection of balls $\mathscr{S}(G, \varepsilon)$ such that G has a vertex cover of cardinality k if and only if $S(G, \varepsilon) = \bigcup \mathscr{S}(G, \varepsilon)$ has an ε-covering of cardinality $k + N$, where N is a constant depending on $\mathscr{S}(G, \varepsilon)$. To simplify notations, we shall refer to $\mathscr{S}(G, \varepsilon)$ and $S(G, \varepsilon)$ simply as $\mathscr{S}$ and S.

4.1 Reduction from vertex cover

Our construction of $\mathscr{S}$ uses two types of balls: rotula balls and ghost balls. **Rotula** balls are balls of radius ε; their centers are called rotulae. They are used in finite sequences of odd length at least 3 that we will call edge gadgets. Two subsequent rotulae of an edge gadget are said to be neighbours of each other. We distinguish two types of rotulae: linking rotulae that are the endpoints of each edge gadget and have only one neighbour, and regular rotulae which are all other rotulae and have two neighbours. See Figure 1b. Besides rotula balls, our construction uses a second type of balls, the **ghost** balls. These balls have radii $\lambda\varepsilon$, $\lambda < 1$; their centers are called ghosts. A single one of these ghost balls constitutes a vertex gadget, which is connected to different edge gadgets through one of their linking rotulae. Thus each linking rotula is associated with one unique ghost, whereas a ghost may be associated with up to 3 linking rotulae depending on the degree of the vertex it was converted from.

Thus, each edge (resp. vertex) in G is converted into an edge gadget (resp. vertex gadget) and we define $\mathscr{S}$ as the collection of all rotula balls and ghost balls resulting from that conversion. At this point, we haven't yet specified the number of rotula balls per edge that we need (only that it should be an odd number) nor the location of rotulae and ghosts. This will be done in Section 4.2 where we build $\mathscr{S}$ from an orthogonal grid drawing of G so that it fulfills the properties below; see Figure 1 for an example.

(i) The erosion of S is exactly the collection of rotulae, $S^{\ominus\varepsilon} = \{c \mid c \text{ is a rotula}\}$.

(ii) Any ball $b \subseteq S^{\oplus\varepsilon}$ contains at most 2 regular rotulae.

(iii) If a ball $b \subseteq S^{\oplus\varepsilon}$ contains 2 regular rotulae, then they are neighbours and b does not contain any other rotula, neither regular nor linking.

(iv) Let c be a regular rotula, c_+ and c_- its two neighbours. There exist two balls $b_+, b_- \subseteq S^{\oplus\varepsilon}$ such that $\{c_+, c\} \subseteq b_+$ and $\{c_-, c\} \subseteq b_-$.

(v) Any ball $b \subseteq S^{\oplus\varepsilon}$ contains at most 3 linking rotulae.

(vi) If a ball $b \subseteq S^{\oplus\varepsilon}$ contains 2 or 3 linking rotulae, then these linking rotulae are associated with the same ghost and b only contains linking rotulae associated with this ghost.

(vii) Let c_g be a ghost. There exists a ball $b \subseteq S^{\oplus\varepsilon}$ that contains all linking rotulae associated with c_g.

(viii) If a ball $b \subseteq S^{\oplus\varepsilon}$ contains both a regular rotula and a linking rotula, then these are neighbours and b does not contain any other rotula, neither regular nor linking.

Henceforth, we shall make no distinction between an edge of G and its conversion into an edge gadget, and likewise for a vertex of G and its corresponding ghost. Recall that an edge gadget $e \in E$ is a finite sequence of rotula balls of odd length at least 3. We denote by $n(e) \geq 1$ the integer such that $2n(e) + 1$ is the length of the edge gadget e. Thus, e has $2n(e) - 1$ regular rotulae. From property (ii) we need at least $\lceil (2n(e) - 1)/2 \rceil = n(e)$ balls in $S^{\oplus\varepsilon}$ in order to cover these regular rotulae. By (iv) there always exists a

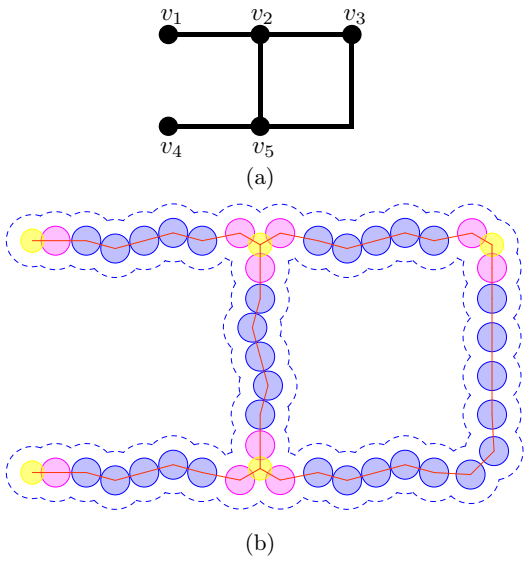


Figure 1: An example of conversion from (a) an orthogonal grid drawing of a graph with 5 vertices and 5 edges to (b) a good collection of balls: blue disks are regular rotulae balls, purple are linking rotulae balls and yellow are ghosts balls. The dilation is bounded by the dashed blue lines. All figures use ghost balls with radius $\lambda\varepsilon$, $\lambda = 0.8$.

collection of $n(e)$ balls in $S^{\oplus\varepsilon}$ which covers these rotulae plus one of the two linking rotulae of e, and that linking rotula can be chosen arbitrarily. Indeed, it suffices to cover pairs of neighbouring rotulae of e with non-overlapping balls in $S^{\oplus\varepsilon}$, while making sure that the linking rotula to cover and its neighbour are one of these pairs (see Figure 2). This gives two possible coverings of regular rotulae of e (one for each linking rotulae) which we shall refer to as **canonical** for e. Furthermore, properties (iii) and (viii) guarantee that any ball containing a regular rotula will only contain rotulae belonging to the same edge gadget. Therefore it is necessary and sufficient to use $n(e)$ balls to cover the regular rotulae of an edge gadget e, and these $n(e)$ balls exclusively cover rotulae of e. However, to entirely cover an edge gadget, one extra ball is required to cover the second linking rotula, for a total of $n(e)+1$ balls. Contrary to the previous $n(e)$ balls, by (v) that extra ball may be shared among several edge gadgets to cover their last linking rotulae. We define $N = \sum_{e\in E} n(e)$. N is the number of balls needed to cover all regular rotulae with canonical coverings.

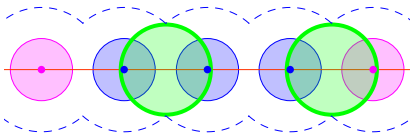


Figure 2: A canonical covering of an edge gadget (green disks). Same color convention as in Figure 1b.

Proposition 3 *If G has a vertex cover $F \subseteq V$, then S has an ε-covering $\mathscr{B}$ with $|\mathscr{B}| = N + |F|$.*

Proof. For each vertex $u \in F$, we use property (vii) and select a ball covering all linking rotulae associated with u. By the vertex cover property, this yields $|F|$ balls that cover at least one linking rotula per edge. Using the appropriate canonical coverings of each edge, we then complete the ε-covering with N more balls to cover the regular rotulae and any remaining linking rotulae. By (i), this collection of balls is an ε-covering since it contains every rotula. □

Proposition 4 *If S has an ε-covering $\mathscr{B}'$, then G has a vertex cover F with $|F| \leq |\mathscr{B}'| - N$.*

Proof. Without loss of generality, we may assume that all balls in $\mathscr{B}'$ cover at least one rotula. Indeed, if a ball b does not cover any rotula, it can be removed from $\mathscr{B}'$ while keeping the property that $\mathscr{B}'$ is an ε-covering. Starting from $\mathscr{B}'$, we first deduce an ε-covering $\mathscr{B}$ of S having the property that it contains one of the two canonical coverings of each edge $e \in E$. For $e \in E$, let $\mathscr{H}(e, \mathscr{B}') = \{b \in \mathscr{B}' \mid \text{b contains a regular rotula of } e\}$. Note that for different edges the $\mathscr{H}(e, \mathscr{B}')$ are disjoint and that $|\mathscr{H}(e, \mathscr{B}')| \geq n(e)$. Given a linking rotula u of $e \in E$, we denote by $\mathscr{C}_e(u)$ the canonical covering that contains u and all the regular rotulae of e. Initializing $\mathscr{B}$ to $\mathscr{B}'$, we then transform $\mathscr{B}$ as follows, replacing each $\mathscr{H}(e, \mathscr{B}')$ according to the following three cases:

- $\bigcup \mathscr{H}(e, \mathscr{B}')$ contains 0 linking rotula. We choose an arbitrary linking rotula u and replace $\mathscr{H}(e, \mathscr{B}')$ with $\mathscr{C}_e(u)$.
- $\bigcup \mathscr{H}(e, \mathscr{B}')$ contains 1 linking rotula u. Then we simply replace $\mathscr{H}(e, \mathscr{B}')$ with $\mathscr{C}_e(u)$.
- $\bigcup \mathscr{H}(e, \mathscr{B}')$ contains both linking rotulae. Then, $|\mathscr{H}(e, \mathscr{B}')| \geq n(e) + 1$. We choose an arbitrary linking rotula u, and let b be a ball containing the other linking rotula but no regular rotula. We replace $\mathscr{H}(e, \mathscr{B}')$ with $\mathscr{C}_e(u) \cup \{b\}$.

Each of these substitutions preserves the ε-covering property and does not increase the cardinality of the resulting collection of balls. Consider the balls of $\mathscr{B}$ that do not contain any regular rotula, $\mathscr{F} = \mathscr{B} \setminus (\cup_{e\in E} \mathscr{H}(e, \mathscr{B}))$. By construction, $|\mathscr{F}| = |\mathscr{B}| - N \leq |\mathscr{B}'| - N$. Let $F = \{u \in V \mid \exists b \in \mathscr{F}, b \text{ contains a linking rotula associated with the ghost } u\}$. We claim that F is a vertex cover of G and that its cardinality satisfies $|F| \leq |\mathscr{F}|$. All $b \in \mathscr{F}$ must contain at least one linking rotula, thus F is empty if and only if $\mathscr{F}$ is empty. In this particular case, the empty set is a vertex cover of G: indeed, G must have no edges because otherwise $\mathscr{B}$ would only cover half of the linking rotulae. Assume now that $\mathscr{F}$ is not empty. By (vi),

any $b \in \mathscr{F}$ yields at most one vertex in F. As for the vertex cover property, recall that $\cup_{e \in E} \mathscr{H}(e, \mathscr{B})$ covers exactly all regular rotulae and one linking rotula per edge. Hence $\mathscr{F}$ must cover the remaining linking rotula of each edge. The definition of F thus ensures it contains at least one endpoint of each edge. □

4.2 Practical construction of $\mathscr{S}$

All that remains is to build a collection $\mathscr{S}$ fulfilling properties (i) to (viii) given G and ε. To do so, we rely on the following result.

Theorem 5 ([2]) *There is a linear time and space algorithm to draw a connected at most cubic graph on an orthogonal grid.*

Given such a drawing of G, we now describe a way to convert it into an appropriate collection of balls. We rely on the orthogonal drawing in Figure 1a as an example. To perform the conversion, we fix the size of the grid to 16ε so that we can fit square blocks of size $8\varepsilon \times 8\varepsilon$ as in Figure 3. There are two different ways in

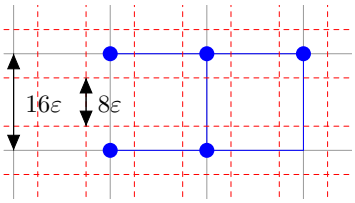


Figure 3: Grid division into blocks. Gray lines represent the grid, dashed red lines are the blocks and the example graph is in blue.

which the blocks meet the graph drawing: the block either contains one vertex or only a portion of one edge. Blocks containing a vertex only vary depending on the number and layout of incident edges. Similarly, blocks containing a portion of edge vary depending on whether the edge bends within the block or not. In each case, we convert the graph drawing covered by the block into a set of balls that satisfies properties (i) to (viii). For blocks containing a vertex, see Figure 4 for the four subcases. Similarly for edges, we have two subcases. However, recall that edge gadgets must have an odd number of rotulae. To achieve this, we use the fact that every edge necessarily crosses at least one block in a straight line and provide an odd and an even conversion for this type of block. The three block conversions are presented in Figure 5.

5 Conclusion

Finding an ε-covering of minimum cardinality is NP-complete. Though the proof presented here relies on a family of shapes with many connected components and whose erosion does not preserve the genus, the result does not depend on it. Indeed, a similar albeit more intricate construction shows that the problem is still hard when restricted to connected shapes that are homotopy equivalent to their erosion.

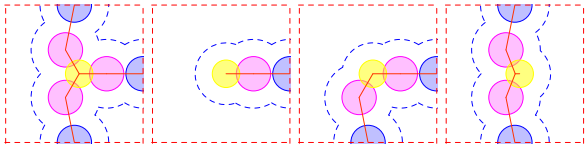


Figure 4: Block conversions for a vertex of degree (a) 3, (b) 1, (c) 2 in a bend and (d) 2 in a straight line. The red dashed square delimit the block. Same color convention as in Figure 1b.

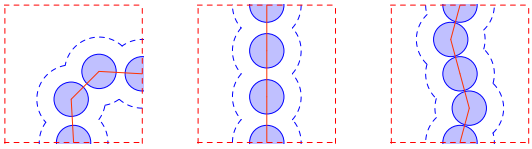


Figure 5: Block conversions for (a) a bent edge, (b) an even and (c) odd straight edge. Same color convention as in Figure 4.

References

[1] G. Bradshaw and C. O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)*, 23(1):1–26, 2004.

[2] T. Calamoneri and R. Petreschi. An efficient orthogonal grid drawing algorithm for cubic graphs. In *Comp. and Combinatorics*, pages 31–40. Springer, 1995.

[3] F. Cazals, T. Dreyfus, S. Sachdeva, and N. Shah. Greedy geometric algorithms for collection of balls, with applications to geometric approximation and molecular coarse-graining. In *Computer Graphics Forum*, volume 33, pages 1–17. Wiley Online Lib., 2014.

[4] F. Cazals and S. Loriot. Computing the arrangement of circles on a sphere, with applications in structural biology. *Computational Geometry*, 42(6):551–565, 2009.

[5] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir. Arrangements of curves in the plane—topology, combinatorics, and algorithms. *Theoretical Computer Science*, 92(2):319–336, 1992.

[6] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[7] B. Miklos, J. Giesen, and M. Pauly. Discrete scale axis representations for 3d geometry. In *ACM Transactions on Graphics (TOG)*, volume 29, page 101. ACM, 2010.

[8] C.-K. Yap. Towards exact geometric computation. *Computational Geometry*, 7(1):3–23, 1997.

On the Separation of a Polyhedron from Its Single-Part Mold*

Dan Halperin† Shahar Shamai†

Abstract

Casting is a manufacturing process where liquid material is poured into a mold having the shape of a desired product. After the material solidifies, the product is pulled out of the mold. We study the case in which the mold is made of a single part and the object to be produced is a three-dimensional polyhedron. We give an algorithm that decides whether a given polyhedron with n facets can indeed be produced that way, and if so indicates how to orient the polyhedron in the mold and in what directions can the product be pulled out without breaking the mold. Our algorithm runs in $O(n \log n)$ time. The best previous algorithm for this problem that we are aware of runs in $O(n^2)$ time. For a convex polyhedron we present a linear-time algorithm.

1 Introduction

Casting is a widely-used manufacturing process, where liquid material is poured into a cavity inside a mold, which has the shape of a desired product. After the material solidifies, the product is taken out of the mold. Typically a mold is used to manufacture numerous copies of a product, in which case we need to make sure that the solidified product can be separated from its mold without breaking it.

The problems that we study here belongs to the larger topic termed *Movable Separability of Sets* by Toussaint [7]. Problems in this area are often exceedingly challenging from a combinatorial- and computational-geometry point of view (see, e.g., [6]). At the same time solutions to these problems are needed in mold design [1], assembly planning [5], and 3D printing to mention a few application areas.

We focus in this paper on a fairly basic movable-separability question. We are given a polyhedron P in $\mathcal{R}^3$ with n facets. We do not make any particular assumptions about the polyhedron besides that it is a closed regular set, namely it does not have dangling edges or facets. The mold is box-shaped and the cavity has the shape of P such that one of P's facets is the top facet of the cavity. See Figure 1 for an illustration in 2D. Once the top facet has been determined, we wish to detect whether there is a direction in which the solidified object could be pulled out of the mold, namely to find a direction that has a positive component in the positive z-direction and such that P could be pulled out of the mold without colliding into the mold. If a direction is found we say that the corresponding top facet is *valid*, and that the polyhedron P is *castable*.

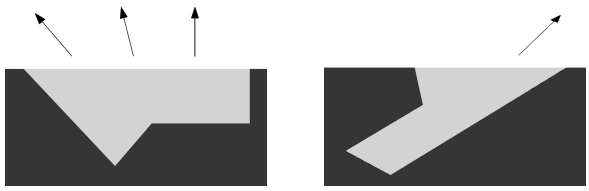


Figure 1: Polygons (light grey) in their molds (darker grey) and valid pull-out directions.

We address two problems:

All Facets Single Direction (AFSD): Determine which facets of P can serve as a valid top facet and for each such facet indicate *one* direction in which P can be pulled out of the mold.

All Facets All Directions (AFAD): Same as above but for each valid facet indicate *all* the directions in which P can be pulled out of the mold.

Why would anyone bother to solve AFAD and not suffice with AFSD? There could be advantages to computing all possible directions. First, it is a more stable solution if there is a continuum of directions rather than a single direction of separation. Second, we could use the availability of many possible directions to impose some direction-related optimality criteria.

Contribution The current algorithms that we are aware of [2, Chapter 4] solve AFSD in $O(n^2)$, and imply an $O(n^2 \log n)$ time solution for AFAD. Both solutions rely on handling each candidate facet to be a top facet separately. Both algorithms, as well as the algorithms that we present below use linear storage space. Our contribution in this paper is an $O(n \log n)$-time algorithm for the AFAD problem. We also present an

*This work has been supported in part by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

†The Blavatnik School of Computer Science, Tel Aviv University, danha@post.tau.ac.il, shasha94@gmail.com.

$O(n)$-time solution to the AFAD problem when the input polyhedron is restricted to be convex.

2 One Structure for All Facets

Instead of handling each candidate top facet separately, as in [2, Chapter 4], we handle all candidate top facets simultaneously using an arrangement of great circles on the unit sphere $\mathcal{S}^2$. Each point p on $\mathcal{S}^2$ represent a direction in $\mathcal{R}^3$—the direction of the vector from the center of $\mathcal{S}^2$ to p. We will use the terms *points* and *directions* on $\mathcal{S}^2$ interchangeably.

Let $F_1, \ldots, F_n$ be the facets of the given polyhedron P. Let $\nu(F_i)$ be the normal to the facet F_i pointing into the polyhedron.

A valid mold is described by F_i as its top facet and a valid pull-out direction $\vec{d'}$. Since we wish to argue about all candidate top facets simultaneously, we will describe a mold by the pair $(F_i, \vec{d})$ which should be interpreted as follows. The top facet of the mold is F_i. To achieve this, the polyhedron needs to be rotated such that F_i becomes the top facet. We apply the same rotation matrix R_i to $\vec{d}$ to obtain a pull-out direction $\vec{d'} := R_i\vec{d}$.

Lemma 1 *The pair $(F_i, \vec{d})$ represents a valid mold and pull-out direction if and only if (i) $\vec{d} \cdot \nu(F_i) < 0$ and (ii) $\forall j \neq i$, $\vec{d} \cdot \nu(F_j) \geq 0$.*

Proof. For the case where F_i is the top facet of P, this fact is proved in Lemma 4.1 in [2]. It remains to notice that the Conditions (i) and (ii) are invariant under rotation. They hold in P's given orientation if and only if they hold when P is rotated such that F_i becomes the top facet. □

Henceforth, we will call a pair $(F_i, \vec{d})$ that obeys the conditions of Lemma 1 a *valid pair*. Notice that the actual pull-out direction is $R_i\vec{d}$, where R_i is the matrix that rotates P such that F_i becomes the top facet, or equivalently such that $\nu(F_i)$ points vertically downwards.

Denote by h_i the closed hemisphere of directions $\vec{d}$ on $\mathcal{S}^2$ for which $\vec{d} \cdot \nu(F_i) \geq 0$, and by $\bar{h}_i$ the open complement hemisphere. Let $\bar{H} = \{\bar{h}_1, \ldots, \bar{h}_n\}$. Let c_i denote the boundary great circle of h_i, and let $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$. Consider the arrangement $\mathcal{A}(\mathcal{C})$ on $\mathcal{S}^2$, namely the subdivision of $\mathcal{S}^2$ induced by the great circles in $\mathcal{C}$ into cells of dimensions $0, 1$ and 2, which we refer to as vertices, edges, and faces respectively.

Definition 1 *The* depth *of a point p on $\mathcal{S}^2$ is the number of hemispheres in $\bar{H}$ in which p is contained.*

Observation 1 *All the points in any fixed cell of the arrangement $\mathcal{A}(\mathcal{C})$ have the same depth.*

The key observation leading to our efficient solution is expressed in the following theorem.

Theorem 2 *The polyhedron P is castable with a single-part mold if and only if the arrangement $\mathcal{A}(\mathcal{C})$ contains a point of depth 1. A cell ξ of depth 1 in $\mathcal{A}(\mathcal{C})$, which is covered by the hemisphere $\bar{h}_i$, represents a mold whose top facet is F_i and each point $\vec{d}$ in ξ represents the valid pull-out direction $R_i\vec{d}$, where R_i is the orthonormal matrix that rotates $\nu(F_i)$ to point vertically down (in the negative z direction).*

Proof. Let ξ be a cell of depth 1 in $\mathcal{A}(\mathcal{C})$ covered by $\bar{h}_i$, and let $\vec{d}$ be a point in ξ. We establish that the pair $(F_i, \vec{d})$ is a valid pair by verifying that the conditions of Lemma 1 above hold for it.

It remains to show that no point in any cell of different depth can represent a valid pull-out direction for any top facet. Consider a cell ψ of depth greater than 1 and a point $\vec{d}$ in it. Let J be the index set of the hemispheres $\bar{h}_i$ that cover ψ: $J = \{i | \vec{d} \in \bar{h}_i\}$. One of the facets $F_j, j \in J$ must serve as the top facet for Condition (i) of Lemma 1 to hold. But then for each of the remaining facets $F_k, k \in J, k \neq j$ Condition (ii) of the lemma is violated. Finally, if a cell has zero depth, then no facet can serve as top facet for any pull-out direction described by points in this cell. □

Remark. Notice that in our setting there cannot be a face of zero depth in $\mathcal{A}(\mathcal{C})$. Similarly, in Lemma 1, Condition (i) follows from Condition (ii).

Our goal is then to compute the cells of $\mathcal{A}(\mathcal{C})$ of depth 1. We first investigate the overall complexity of all these cells.

Proposition 3 *The overall complexity of the depth-1 cells in $\mathcal{A}(\mathcal{C})$ is $O(n)$.*

Proof. We refer to the horizontal great circle of $\mathcal{S}^2$ as the *equator*. The equator splits $\mathcal{S}^2$ into the open lower hemisphere and the open upper hemisphere. We will handle each of these three parts of $\mathcal{S}^2$ separately.

Assume temporarily that no hemisphere h_i has the equator as it bounding great circle.

We start with the upper hemisphere of $\mathcal{S}^2$. Centrally project the intersection of the upper hemisphere with each of the hemispheres $\bar{h}_i$ onto the plane $z = 1$. For each $\bar{h}_i$ we obtain a halfplane $\bar{g}_i$. Let g_i denote the complementary closed half plane, and ℓ_i the line bounding each of them. Let G, $\bar{G}$, and $\mathcal{L}$ be the corresponding sets of n elements each. See Figure 2 for an illustration.

We call a vertex v of the arrangements $\mathcal{A}(\mathcal{L})$, which is the intersection point of two distinct lines in $\mathcal{L}$, a *configuration*. We say that such a vertex v is in conflict with the halfplane $\bar{g} \in \bar{G}$ if $v \in \bar{g}$. We are interested in counting all the vertices of $\mathcal{A}(\mathcal{L})$ that have

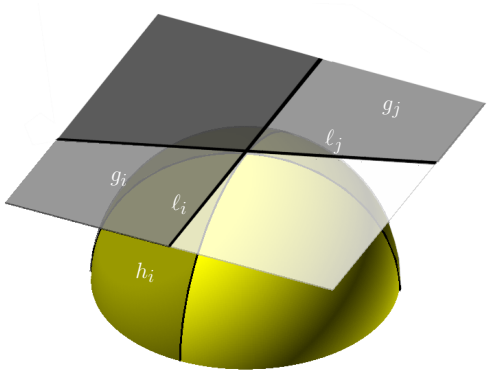


Figure 2: Central projection of hemispheres onto the plane $z = 1$.

exactly one conflict. These will correspond to vertices of depth 1 in the upper hemisphere of $\mathcal{A}(\mathcal{C})$; more generally the number of conflicts of a configuration is the depth of the vertex. Let $N_k(\mathcal{L})$ denote the number of vertices of depth k, and $N_{\leq k}(\mathcal{L})$ denote the number of vertices of depth at most k. Let $N_k(n)$ (resp. $N_{\leq k}(n)$) denote the maximum $N_k(\mathcal{L})$ (resp. $N_{\leq k}(\mathcal{L})$) over all sets $\mathcal{L}$ of n lines. To bound the maximum number of vertices of depth 1 we use the Clarkson-Shor framework [3], which asserts that

$$N_{\leq k}(n) = O(k^d N_0(n/k)) \ ,$$

where d is the number of objects in the set that define a configuration. In our case $d = 2$ and $k = 1$.

What we still need to bound is $N_0(n)$, namely the maximum number of configurations with no conflicts in any set $\mathcal{L}$ of n lines. However, this is easy: these are the vertices on the boundary of the intersection of the halfplanes in G. Therefore $N_0(n) = n$. It follows that $N_{\leq 1} = O(n)$ as asserted.

Analogous arguments apply to the lower hemisphere.

Next, we consider $\mathcal{A}(\mathcal{L})$ restricted to the equator as a one-dimensional arrangement consisting of vertices and arcs, namely 0- and 1-dimensional cells. It is not difficult to see that the complexity of the cells of depth 1 in this arrangement is $O(1)$. There can be at most four such cells—in a degenerate situation where there are two pairs of complementary half-circles, which in turn induce two pairs of antipodal 0-cells of depth 1.

We now relax the assumption that there are no hemispheres whose bounding circle is the equator. Notice that the introduction of such hemispheres does not affect the asymptotic upper bounds derived so far. There is the possible introduction of a constant number of extra vertices of depth 1 on the equator. □

Can one compute these faces efficiently? The Clarkson-Shor framework [3] also gives a randomized algorithm to compute these faces in expected near-linear time. However, we will next describe a simple deterministic algorithm running in $O(n \log n)$ time for that purpose.

3 The Algorithm

We now compute all the valid pairs $(F_i, \vec{d})$, by computing all the cells of depth 1 in the arrangement $\mathcal{A}(\mathcal{C})$. We handle each of the upper hemisphere, the equator, and the lower hemisphere separately.

For the upper hemisphere we use the projection described above, the set $\bar{G}$ of half-planes, and the set $\mathcal{L}$ of their bounding lines. Notice that there is one-to-one correspondence between the points on the plane $z = 1$ and the points of the open upper hemisphere. Thus once we find the cells of depth 1 in $\mathcal{A}(\mathcal{L})$ we immediately get the desired pairs. Hence we focus on the plane $z = 1$.

We subdivide the set $\bar{G}$ into the disjoint union of three subsets: $\bar{G}_1, \bar{G}_2$, and $\bar{G}_3$, having their half-planes above their bounding line, below their bounding line, or having a vertical bounding line, respectively. Let $\mathcal{L}_i$ be the set of bounding lines of $\bar{G}_i, i = 1, 2, 3$. Let λ_i be the number of lines in $\mathcal{L}_i$.

Recall the definition of levels in arrangements, which we will need below. Given a set $\mathcal{L}$ of lines in the plane, we define the *level* of a point p in the plane to be the number of lines in $\mathcal{L}$ strictly below p [4]. We say that an edge e in the arrangement $\mathcal{A}(\mathcal{L})$ is at level k if there is a point in the interior of e at level k (and hence all interior points of e are at level k). The k-level of the arrangement $\mathcal{A}(\mathcal{L})$ is the closure of the union of edges of $\mathcal{A}(\mathcal{L})$ that are at level k. See Figure 3.

Back to the depth-1 faces, we handle the three sets separately and then merge the results. We start with $\bar{G}_1$. If we restrict our attention to the half-planes in $\bar{G}_1$, the points of depth k in the plane correspond exactly to points in the plane at level k. Thus our goal is to compute the points at levels 0 and 1. (We need level 0 as well since we will later merge this result with the results for the other sets.) We use divide-and-conquer on the lines in $\mathcal{L}_1$. Assume that we have already computed the points at levels 0 and 1 for each of the arrangements $\mathcal{A}(L_1^a)$ and $\mathcal{A}(L_1^b)$, where $\mathcal{L}_1^a$ and $\mathcal{L}_1^b$ are two disjoint subsets of $\mathcal{L}_1$ of size $\lambda_1/2$ each. The complexity of level 0 in each is obviously $O(\lambda_1)$ and so is the complexity of level 1 (this is well known; it follows for example as a special case of Proposition 3 above). What we actually compute are the levels 0 and 1 of the arrangements, namely two polygonal lines rather than a planar subdivision. (This will no longer be true when we merge the final result for $\bar{G}_1$ with the results for the other subsets of $\bar{G}$.) Therefore we can easily carry out the current merge step by projecting the breakpoints of both levels in each

arrangement onto the x-axis and merging these lists of projected points into one list. We then handle in constant time the slab above each interval between two consecutive breakpoint projections. This way we obtain the level 0 and level 1 faces of the arrangement in time $O(\lambda_1 \log \lambda_1)$.

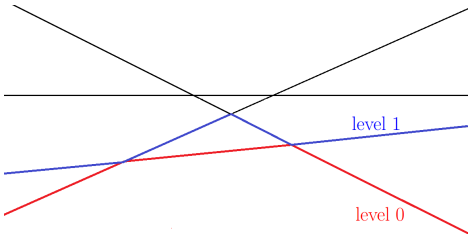


Figure 3: Levels in an arrangement of lines.

We operate analogously on the set $\bar{G}_2$, and merge the results with those already computed for $\bar{G}_1$ in a merge step that is similar to the merge step we have just described for $\bar{G}_1$, still keeping the faces of levels 0 and 1. Now, however, we need to maintain a planar subdivision, say by using a DCEL [2, Chapter 2]. From this point on, we also record with each cell of depth 1 which is the half-plane that covers it.

Next we compute the relevant faces for $\bar{G}_3$. This is easier and can be carried out in $O(\lambda_3)$ time , resulting in only a constant number of faces. We omit the details here. We finally merge this constant size result with the result of the preceding step in $O(n)$ time. This concludes the handling of the upper hemisphere of $\mathcal{S}^2$.

The lower hemisphere is handled analogously. Computing the depth-1 cells on the equator is simpler yet and can be carried out in $O(n)$ time by an incremental algorithm adding the circular arcs describing forbidden directions one after the other. At any time during the algorithm there is a constant number of arcs of depth 0 and 1.

At all times our algorithm does not require more than $O(n)$ storage as we keep a constant number of linear lists (describing levels 0 and 1 of certain arrangements), or a planar subdivision of linear size. In summary

Theorem 4 *Given a polyhedron P with n facets, we can find in $O(n \log n)$ time all the valid pairs $(F_i, \vec{d})$, where P can be pulled in direction $R_i\vec{d}$ out of a mold having F_i as a top facet, and R_i is the matrix that rotates P such that F_i becomes the top facet. The algorithm requires linear space.*

Finally we state our more efficient algorithm for the analogous AFAD problem for an arbitrary polygon in the plane. The proof is omitted for lack of space.

Theorem 5 *Given a polygon Q with n edges in the plane, we can find in $O(n)$ time all the valid pairs of top edge and pull-out directions. The algorithm uses only constant-size working storage.*

4 Casting Convex Polyhedra

In this section we show how to determine the castability of a *convex* polyhedron more efficiently, still solving the AFAD problem for this case.

We say that two facets of the input polyhedron P are neighbors if their closures intersect in an edge. Denote by $\mathcal{M}_i$ the set of neighbors of the facet F_i, by m_i the cardinality of this set, and by J_i the index set of the facets in $\mathcal{M}_i$. The efficient algorithm stems from the following observation (proof omitted).

Proposition 6 *For a convex polyhedron, the pair $(F_i, \vec{d})$ represents a valid mold and pull-out direction iff (i) $\vec{d} \cdot \nu(F_i) < 0$ and (ii) $\forall j \in J_i, \vec{d} \cdot \nu(F_j) \geq 0$.*

The algorithm proceeds by fixing a face F_i, computing its edge-neighboring facets and computing the intersection of allowable directions (half-planes on $z = 1$) in $O(m_i)$ time, using the order of the neighbors along the boundary of f. We repeat the procedure for each facet of P. Notice that m_i is in fact the number of edges on the boundary of F_i. The overall cost $O(m_i)$ over all candidate top facets F_i is $O(n)$ by Euler's formula. In summary

Theorem 7 *The AFAD problem for a* convex *polyhedron P with n facets is solvable in time $O(n)$.*

References

[1] H. Ahn, M. de Berg, P. Bose, S. Cheng, D. Halperin, J. Matoušek, and O. Schwarzkopf. Separating an object from its cast. *Computer-Aided Design*, 34(8):547–559, 2002.

[2] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

[3] K. L. Clarkson and P. W. Shor. Application of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[4] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, 2nd edition, 2004.

[5] D. Halperin, J. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26(3-4):577–601, 2000.

[6] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. *Discrete & Computational Geometry*, 12:367–384, 1994.

[7] G. Toussaint. Movable separability of sets. In *Computational Geometry*. North-Holland Publishing Company, 1985.

Covering points with rotating polygons

Carlos Alegría-Galicia * David Orden † Leonidas Palios ‡ Carlos Seara § Jorge Urrutia ¶

Abstract

We study the problem of rotating a simple polygon to contain the maximum number of elements from a given point set. We consider variations of this problem where the rotation center is a given point or lies on a line segment, a line, or a polygonal chain.

1 Introduction

Given a simple polygon P, the *Polygon Placement Problem* consists in finding a function τ such that a *placement* $\tau(P)$ satisfies a certain property, for τ combining certain allowed types of movements. The oldest problem of this family we are aware of was studied in the early eighties by Chazelle [5], who given two polygons P and Q explored the problem of finding, if it exists, a placement $\tau(P)$ that contains Q using translation and rotation.

The most recent contribution to these problems is due to Barequet and Goryachev [3]. Among other results, for a point set S, a simple polygon P, and τ a composition of translation and rotation, they show how to compute a *maximum cover placement* for P, that is, a placement $\tau(P)$ containing the maximum number of points of S. For n and m being the sizes of S and P respectively, their algorithm runs in $O(n^3m^3\log(nm))$ time and $O(nm)$ space.

Although translation-only problems have also being considered [1], to the best of our knowledge there are no previous results where τ is only a rotation[1]. In this paper we thus study the following *Maximum Cover under Rotation (MCR)* problems:

Problem 1 (Fixed MCR) *Given a point r in the plane, compute an angle $\theta \in [0, 2\pi)$ such that, after counterclockwise rotating P by θ around r, the number of points of S contained in P is maximized.*

Problem 2 (Segment Restricted MCR) *Given a line segment ℓ, compute a point r on ℓ and an angle $\theta \in [0, 2\pi)$ such that, after counterclockwise rotating P by θ around r, the number of points of S contained in P is maximized.*

Applications of polygon placement problems include global localization of mobile robots, pattern matching, and geometric tolerance (see the references in [3]). Rotation-only versions arise in robot localization using a rotating camera [7] or quality control of objects manufactured around a vertical axis.

We show that Problem 1 is 3SUM-hard (an $o(n^{2-\varepsilon})$-time solution for it implies an affirmative answer to the open question of whether an $o(n^{2-\varepsilon})$-time algorithm for 3SUM exists [6]) and present two algorithms to solve it: one requiring $O(nm\log(nm))$ time and $O(nm)$ space, the other taking $O((n+k)\log n + m\log m)$ time and $O(n+m+k)$ space, where $k = O(nm)$ is the number of events. We also describe an algorithm that solves Problem 2 in $O(n^2m^2\log(nm))$ time and $O(n^2m^2)$ space. This algorithm can be easily extended to solve variations of Problem 2 where r lies on a line or a polygonal chain.

2 Fixed MCR (Problem 1)

Let c_p be the circle with center r and radius $|\overline{rp}|$, where p is a point in S. If instead of rotating P counterclockwise we rotate S in clockwise direction, c_p is the curve described by p during a 2π rotation around r. The endpoints of the circular arcs resulting from intersecting P and c_p mark the rotation angles where p enters (*in-event*) and leaves (*out-event*) the polygon P. In the worst case, the number of such events per element of S is $O(m)$, for a total of $O(nm)$ if we consider all the points in S. See Figure 1.

*Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, calegria@uxmcc2.iimas.unam.mx.

†Departamento de Física y Matemáticas, Universidad de Alcalá, Spain, david.orden@uah.es. Research supported by MINECO Projects MTM2014-54207 and TIN2014-61627-EXP and by TIGRE5-CM Comunidad de Madrid Project S2013/ICE-2919.

‡Department of Computer Science and Engineering, University of Ioannina, Greece, palios@cs.uoi.gr.

§Departament de Matemàtiques, Universitat Politècnica de Catalunya, Spain, carlos.seara@upc.edu. Research supported by projects Gen. Cat. DGR 2014SGR46, MINECO MTM2015-63791-R.

¶Instituto de Matemáticas, Universidad Nacional Autónoma de México, urrutia@matem.unam.mx. Research supported in part by MTM2006-03909 (Spain) and SEP-CONACYT of México, Proyecto 80268.

[1]Existing results where τ is a composition of either rotation, translation, and scaling, reduce the search space complexity by only considering placements where a constant number of points from S lie on the boundary of P (see for example references [3] and [4] for algorithms based respectively, on two-point and one-point placements). Rotation-only adaptations of these results would not allow the rotation center to be fixed or restricted to lie on a given curve and therefore, cannot be applied to the problems we deal with in this paper.

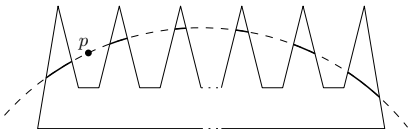


Figure 1: A comb-shaped simple polygon can generate $\Omega(m)$ in- and out-events per point in S.

2.1 A 3SUM-Hard reduction

We show next that Problem 1 is 3SUM-hard by a reduction from the Segments Containing Points problem that was proved to be 3SUM-hard by Barequet and Har-Peled [2].

Problem 3 (Segments Containing Points) *Given a set A of n real numbers and a set B of $m = O(n)$ pairwise-disjoint intervals on the real line, is there a real number u such that $A + u \subseteq B$?*

Theorem 1 *Fixed MCR is 3SUM-hard*[2].

Proof. Let I be an interval of the real line that contains the set A of points and the set B of intervals of an instance of the Segments Containing Points problem. Wrap I on a circle C whose perimeter has length at least twice the length of I. This effectively maps the points in A and the intervals in B into a set A' of points and a set B' of intervals on C.

Clearly, finding a translation (if it exists) of the elements of A such that $A + u \subseteq B$, is equivalent to finding a rotation of the set of points A' such that all of the elements of A' are mapped to points contained in the intervals of B'. To finish our reduction, construct a polygon as shown in Figure 2.

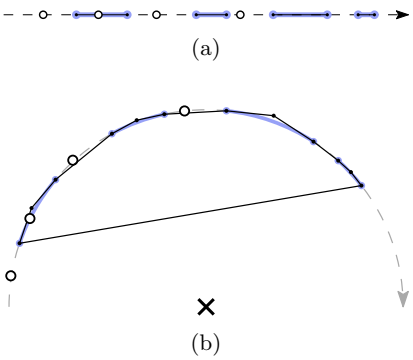


Figure 2: Wrapping I from (a) the real line to (b) a circle C. Intervals forming B and B' are highlighted with blue. Elements of A and A' are represented by white points. Additional vertices forming the polygon are the intersection points between the tangents to C at the endpoints of each interval in B'.

□

[2]The proof of this theorem is based on the proof of Theorem 4 from Barequet and Har-Peled [2].

2.2 An $O(nm\log(nm))$ algorithm.

By Theorem 1 it is unlikely that we could solve Problem 1 in less than quadratic time. We outline now our best solution.

1. Intersect rotation circles. Compute the intersection points of c_p and P, for every p in S.

2. Compute the sequence of events. Choose a common reference and translate every intersection point into a rotation angle in S^1. Sort all the events by increasing angular order into an event sequence, and determine if they define in- or out-events (see Figure 3). Note that, for each element p_j of S, we obtain a sequence of sorted intervals $\mathcal{I}_j = \{I_{j,1}, \ldots, I_{j,i_j}\}$ that determine the rotation angles for which p_j belongs to P.

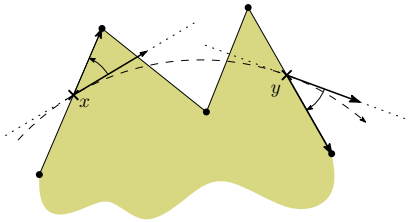


Figure 3: An in-event at x (left turn), and an out-event at y (right turn).

3. Compute the angle of maximum coverage. Using standard techniques, we can now perform a sweep on the set obtained by joining all of the intervals in $\mathcal{I}_1 \cup \cdots \cup \mathcal{I}_n$.

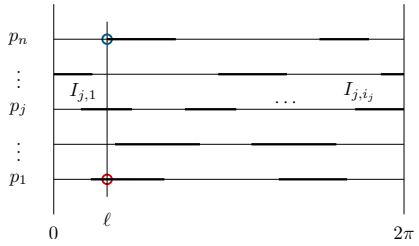


Figure 4: The events sequence and the sweeping line at angle θ. Highlighted with a red circle, the intersection of line ℓ with an interval corresponding to p_1 (p_1 is inside P). Highlighted with a blue circle, the intersection of line ℓ with one of the endpoints of an interval corresponding to p_n (an in-event).

During the sweeping process, we keep a counter containing the number of points of S in P. If an in-event or an out-event occurs, the counter is increased or decreased by one, respectively. At the end of the sweeping process, we report the angular interval(s) where the count is maximized.

Since the complexity of our algorithm is dominated by items 1 and 2, which take $O(nm\log(nm))$ time:

Theorem 2 *The Fixed MCR problem can be solved in $O(nm\log(nm))$ time and $O(nm)$ space.*

2.3 A more efficient algorithm.

Performing a plane sweep using a circular sweepline outwards from the rotation center r, it is possible to intersect P and the set of rotation circles in a more efficient way. The idea is to maintain a list of the edges intersecting the sweepline, ordered by appearance while the sweepline is traversed in clockwise direction around r. Using the same technique shown in Figure 3, the edges are labeled as defining in- or out-events. The algorithm is outlined next.

1. Normalize P. In the following steps, we consider P to have no edges intersecting a rotation circle more than once. This can be guaranteed by performing a preprocessing step on P: For every edge $e = \overline{ab}$ of P, let p_e be the intersection point between the line ℓ containing e and the line perpendicular to ℓ passing through r. If p_e belongs to the relative interior of e, subdivide it into the edges $\overline{ap_e}$ and $\overline{p_eb}$. In the worst case, each edge of P gets subdivided in two parts. See Figure 5.

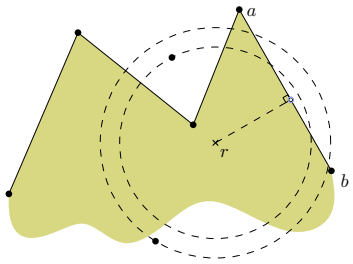


Figure 5: Splitting an edge of P.

2. Process a vertex of P**.** When the sweepline stops at a vertex of P, we update the ordered list of edges intersected by the sweepline.

3. Compute the intervals sequence for each element of S**.** When the sweepline reaches a point p_j in S, we are ready to compute the sequence $\mathcal{I}_j$ of sorted intervals of p_j. It suffices to walk along the ordered list of edges intersected by the sweepline, and compute the corresponding angles clockwise from the ray emanating from r towards p_j.

4. Construct the events sequence. Since for each point in S we have computed the corresponding sequence of sorted intervals, all we need to do is to merge these (at most n) sequences into a complete sequence of events. We do that in a balanced fashion as in the merge sort algorithm.

The normalization process takes $O(m)$ time. Sorting the points in S and the vertices of P by distance from r takes $O(n\log n)$ and $O(m\log m)$ time, respectively. The ordered list of edges intersecting the sweepline is maintained in a balanced binary search tree, so we can process all the vertices of P in $O(m\log m)$ time. On the other hand, processing all the points in S takes $O(k)$ time (recall that k denotes the total number of in- and out-events in a Fixed MCR problem). Finally, merging the $O(n)$ sequences of sorted intervals takes $O(k\log n)$ time from which in $O(k)$ time we obtain a solution. In total, the time complexity of the algorithm is $O(n\log n + m\log m + k\log n)$ time. The space complexity is $O(n+m+k)$. We have thus proved:

Theorem 3 *The Fixed MCR problem can be solved in $O((n+k)\log n + m\log m)$ time and $O(n+m+k)$ space.*

3 Segment Restricted MCR (Problem 2)

Let $\ell = \overline{ab}$ be the line segment restricting the position of the rotation center r. Our approach to solve Problem 2 is to characterize, for each p in S, the intersection between P and the rotation circle c_p while r moves along ℓ from a to b. For each edge $e = \overline{uv}$ of P, we parametrize the intersection between c_p and e using a function $\omega = f(t)$, for ω being the clockwise angle shown in Figure 6, and t the y-coordinate of r. For simplicity, we assume that a lies on the origin $(0,0)$ and b on the positive y-axis.

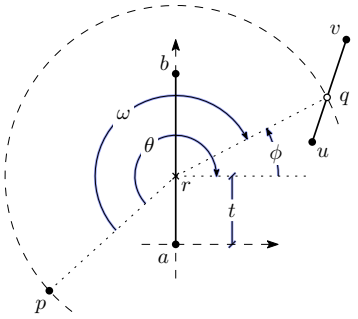


Figure 6: Parametrizing the intersection between c_p and $\overline{uv}$ while r moves along $\overline{ab}$.

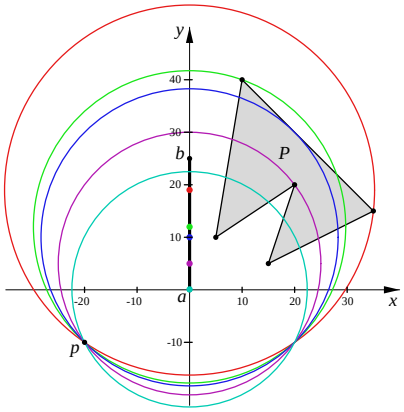
If we consider clockwise and counterclockwise angles being positive and negative respectively, we have from Figure 6 that $\omega = \theta + \phi$. The angle θ can be easily computed in terms of t. By equating the distances from r to p and q and invoking $z = \tan\phi$, we get an equation of the form

$$Az^2 + Bt^2 + Ctz^2 + Dt^2z + Etz + Ft + Gz + H = 0, \quad (1)$$

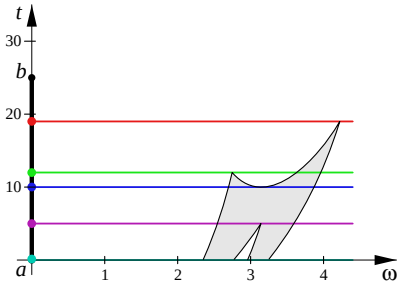
where $A, \ldots, H$ are constants depending on the coordinates of p, u, and v. By resolving Equation (1) for t we obtain

$$t = \frac{f(z) \pm \sqrt{g(z)}}{h(z)}, \quad (2)$$

where $f(z)$, $g(z)$, and $h(z)$ are polynomials of degrees 2, 4, and 1, respectively. The motion of r along ℓ thus corresponds to a set of points (t, ω) for which p belongs to P. These points form a set of simple regions in the t-ω plane which are bounded by $O(m)$ curves. Any pair of such regions have disjoint interiors, whereas their boundaries may intersect at most at a common vertex. See Figure 7.



(a)



(b)

Figure 7: (a) A Segment Restricted MCR instance for a point p in S and (b) its corresponding t-ω diagram, where the ω axis is measured in radians.

By processing all the points in S we end up with a set of $O(nm)$ regions bounded by $O(nm)$ curves in the t-ω plane. From Equation (2) we can show that any two such curves intersect at most a constant number of times, for a total of $O(n^2m^2)$ intersection points in the worst case. Using standard techniques, in $O(n^2m^2 \log(nm))$ time the arrangement of all these regions can be computed, and the dual graph of the resulting arrangement can be traversed looking for the sub-region of maximum depth. Any point in this sub-region determines a position of r and a rotation angle ω that constitute a solution to the problem. In summary we have:

Theorem 4 *The Segment Restricted MCR problem can be solved in $O(n^2m^2 \log(nm))$ time and $O(n^2m^2)$ space.*

Note that Problem 2 can also be solved in $O(n^2m^2 \log(nm))$ time even when r is restricted to lie on a line L: Compute the Voronoi diagram of S and the vertices of P, and apply the algorithm we just described to a segment of L containing all the intersection points of L and the Voronoi edges. Moreover, if we restrict r to lie on a polygonal chain with s segments, we can trivially obtain the optimal placement of P using $O(sn^2m^2 \log(nm))$ time. In both cases the space complexity is $O(n^2m^2)$.

4 Concluding remarks

We studied the problem of finding a rotation of a simple polygon that covers the maximum number of points from a given point set. We described algorithms to solve the problem when the rotation center is fixed, or lies on a line segment, a line, or a polygonal chain. Without much effort our algorithms can also be applied when the polygon has holes, and can be easily modified to solve minimization versions of the same problems.

References

[1] G. Barequet, M. T. Dickerson, and P. Pau. Translating a convex polygon to contain a maximum number of points. *Computational Geometry: Theory and Applications*, Vol. 8(4), (1997), 167–179.

[2] G. Barequet and S. Har-Peled. Polygon containment and translation min-hausdorff-distance between segment sets are 3SUM-hard. *International Journal of Computational Geometry & Applications*, Vol. 11(4), (2001), 465–474.

[3] G. Barequet and A. Goryachev. Offset polygon and annulus placement problems. *Computational Geometry: Theory and Applications*, Vol. 47(3, Part A), (2014), 407–434.

[4] D. Matthew and S. Daniel. Optimal placement of convex polygons to maximize point containment. *Computational Geometry: Theory and Applications*, Vol. 11(1), (1998), 1–16.

[5] B. Chazelle. *Advances in Computing Research*, Vol. 1. Chapter: The polygon containment problem, 1–33, JAI Press, (1983).

[6] A. Gajentaan and M.H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, Vol. 5(3), (1995), 165–185.

[7] H. Ishiguro, M. Yamamoto, and S. Tsuji. Omni-directional stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14(2), (1992), 257–262.

Trash Compaction

Hugo Akitaya* Greg Aloupis* Maarten Löffler† Anika Rounds*

Abstract

Let P be a set of n objects on a square grid. A *push* is a transformation of P that involves sweeping a horizontal or vertical line by one unit, starting from the hull of P, displacing objects in the direction of the sweep. For example, when pushing to the right, all the leftmost objects are displaced one unit to the right. This in turn displaces other objects further right. Given P, we want to find a sequence of pushes that will produce a rectangle of a given height and width. We show that deciding whether a square can be produced is NP-hard, but it takes polynomial time to decide if a rectangle of height 2 can be produced.

1 Introduction

There is a rich history in computational and combinatorial geometry on packing unit squares as tightly as possible into various domains. In 1975, Erdös and Graham asked how many unit squares may be packed into a square of given dimensions, allowing arbitrary rotations [6]. This sparked a string of results [8], as well as new questions and variations. Of particular interest is the version where unit squares are not allowed to rotate [7]. The problem has recently been shown to be NP-complete when the domain is a rectilinear polygon with half-integer side lengths [5, 10].

Pushing objects is a vital task in certain motion planning settings with obstacles. Dhagat and O'Rourke first considered the problem of pushing square obstacles on a grid [4]. Many versions of the problem are NP-hard [3], in particular various popular games involving pushing blocks. Recently, the motion planning community has shown interest in controlling configurations of objects using only global interactions, motivated from swarm robotics [1, 2].

In this paper, we consider the *trash compaction* problem: given a set P of n objects (pieces of trash) at integer coordinates in the plane, can we we push them into a more compact configuration using only axis-aligned global *push* operations? In each of the four cardinal directions, we can perform a *push* operation on P. This involves sweeping a line in the given direction by one unit. Any object swept by the line is displaced, thus moving to the next integer coordinate. If another object occupies that coordinate, it is also displaced by one unit, etc. An example of a *left* push is shown in Figure 1. We focus on pushing objects into rectangular configurations. Trivially, this cannot always be done, as shown in Figure 1c.

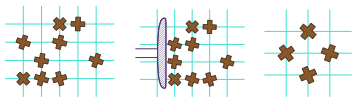


Figure 1: (a) Configuration of objects. (b) Configuration after a left push. (c) Configuration that cannot be pushed into an axis-aligned square.

1.1 General observations

There are some characteristics of this problem that we use in the proof of the main results in this paper. Due to space constraints, some proofs are omitted; they can be found in the full version.

If we are interested in pushing a configuration into a rectangle with dimensions $k \times \frac{n}{k}$, then trivially we require that k divides n.

Observation 1 *Suppose that a push causes j objects to be displaced within a row. This is equivalent to moving the first object to the closest available empty space in the row, j positions away.*

Observation 2 *Pushing horizontally* (resp. *vertically) cannot decrease the number of objects in any column* (resp. *row).*

Observation 3 *If any column contains more than k objects, or any row contains more than $\frac{n}{k}$ objects, then it is not possible to produce a $k \times \frac{n}{k}$ rectangle.*

In general, pushing horizontally then vertically is not equivalent to pushing vertically then horizontally (consider switching the last two pushes in Figure 2).

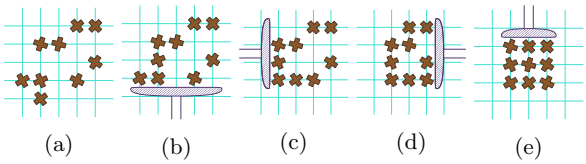


Figure 2: Four pushes to create a square.

*Tufts University, Medford, MA, USA
†Utrecht University, The Netherlands

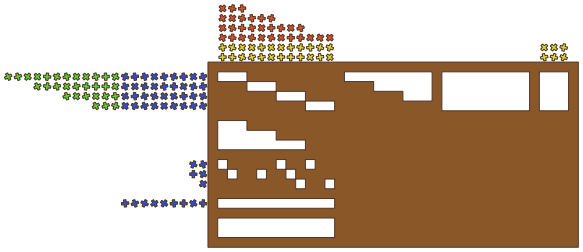
Lemma 1 *Pushing left then right is equivalent to pushing right then left.*

Lemma 2 *Suppose a configuration occupies exactly k rows (not necessarily consecutive). Then if a rectangle of dimensions $k \times \frac{n}{k}$ can be created, one way to do so is to push left until the configuration occupies $\frac{n}{k}$ consecutive columns, then push down until the configuration occupies k consecutive rows.*

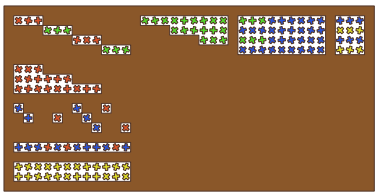
2 Pushing into any rectangle

We show hardness of a restricted version of the problem, where only top and left pushes are allowed. Then we extend the result for the original question. Our reduction is from Exact-Hitting-Set (which is NP-hard [9]): Given a set $S = \{1, \ldots, n\}$, and m sets $S_1, \ldots, S_m$, each a subset of S, decide if there is a subset $S' \subset S$ of exactly k elements, such that each S_i has exactly one element in common with S'.

Construction. We build a rectangle with dimensions $\Theta(nm) \times \Theta(n+m)$. We fill this rectangle with objects, except for several empty *holes*. We then place the same number of objects as is needed to fill all the holes, above and to the left of the construction, in such a way that any sequence of top and left pushes that correctly fills the holes must correspond to a solution for Exact-Hitting-Set. To make the proof easier to follow, we color the objects that start outside the main rectangle. Figure 3 is an example for n=4, m=3, k=2. S_1={1, 3, 4}, S_2={1, 2, 3}, S_3={3, 4}.



(a) n=4, m=3, k=2. S_1={1, 3, 4}, S_2={1, 2, 3}, S_3={3, 4}



(b) S'={2, 4}

Figure 3: Reduction from Exact-Hitting-Set. For a larger copy with labeled regions, see the full version.

For our description, we use the convention described in Observation 1. There are eight main regions of the rectangle in which we insert holes. The upper-left *Staircase* region contains n horizontal holes, each of length m. To the right of the Staircase are three big holes: *Green-triangle*, *Green-overflow* and *k-check*. To the bottom of the Staircase is a big hole called *Red-triangle*, followed by a region of small holes called *Hit-check* and two big holes: *Red-overflow* and *Yellow-buffer*. The Staircase is nm wide and n tall. We consider that S is arbitrarily ordered and the i-th element ($i \in \{1, \ldots, n\}$) is represented by a hole of height 1 in the i-th row of the Staircase ranging from column $(i-1)m+1$ to im. The Green-triangle and Red-triangle are $(n-1)m$ wide and $n-1$ tall and the difference of the number of empty columns between adjacent rows is m. The Green-overflow, k-check, Red-overflow and Yellow-buffer are rectangular holes of dimensions $n \times (n-k+1)m$, $n \times m$, $1 \times nm$ and $k \times nm$, resp. In the Hit-check region, make an $n \times m$ matrix of holes encoding the sets S_j. Each row represents a set. One element is represented by m columns. If the i-th element of S is in S_j, row i has a 1×1 hole in column $(i-1)m+j$. Outside the rectangle, right above the Staircase, place a $k \times nm$ rectangle of yellow objects. Right above, place n rows of red objects, each containing m less objects than the previous (starting with nm). Above the k-check region, place a $k \times m$ rectangle of yellow objects. To the left of the Staircase, place $n \times (n-k+1)m$ blue objects. To the left of the i-th row of blue objects, place $nm-(i-1)m$ green objects. To the left of the i-th row of the Hit-check region add $|S_i| - 1$ blue objects. Finally, to the left of the Red-overflow, add $1 \times (n-k+1)m$ blue objects.

Correctness. First consider that the hitting set problem has a positive solution S'. We convert S' into a sequence of moves that result in a rectangle. For $i \in \{1, \ldots, n\}$, if the i-th element of S is in S', left-push m times then down-push once. If the i-th element of S is not in S', down-push once then left-push m times. Left-push $(n-k+1)m$ times then down-push k times. This sequence will fill the holes in the Staircase corresponding to elements in S' with green objects and other holes with red objects. The Green-triangle and Red-triangle are filled with green and red objects respectively. The Green-overflow will be filled mostly with blue objects. Since $|S'| = k$, exactly $n-k$ rows of the Green-overflow will contain m green objects and, therefore, the k-check hole is filled with $n-k$ rows of blue and k rows of yellow objects. Because S' hits each subset exactly once, the objects to the left of the Hit-check and Red-Overflow will fill the holes that are not filled with red objects. Thus, this sequence compacts all objects into a rectangle.

Now, consider that there exists a sequence that results in a rectangle. The k-check hole can only be filled by yellow and blue objects and, since yellow objects can only be pushed into the rectangle after all red objects, there must be $n-k$ rows completely filled with blue objects in the k-check hole. That implies that each hole in the Staircase can only be filled with objects of the same color (either green or red) and that exactly k such holes are filled with green objects. Holes in the Hit-check region can only be filled with red or blue objects. A hole in such a region is filled with red objects only if the hole directly above it in the Staircase is filled with green objects. Since there are $|S_i|-1$ blue objects to the left of the i-th line, exactly one hole in each line must be filled with a red object. Therefore, the set S' of holes filled with green objects in the Staircase corresponds to a solution to the Exact-Hitting-Set instance.

Four-sided trash compaction. The reduction can be adapted to the model that allows pushes from all four cardinal directions. We remove one object from the top-left of the rectangle and add a *blocker* object at the bottom-right of the construction, which will prevent us from pushing from the right or bottom, because it would cause a row or column to become too long. However, once we resolve all pushes from the top and left, one top push and one left push incorporate the *blocker* filling the gap at the top-left corner.

Theorem 3 *Given a configuration P of n objects, and two integers w and h, deciding whether P can be pushed into a $w \times h$ configuration is NP-hard.*

Corollary 4 *Deciding if a configuration can be pushed into a $\sqrt{n} \times \sqrt{n}$ configuration is NP-hard.*

3 Pushing into a rectangle of height 2

It is trivial to decide if P can be pushed into a single row. This can be done if and only if each column contains at most one element. To decide if P can be pushed into a rectangle of height 2 requires more effort, even if P initially occupies only three rows. Let r_1, r_2, and r_3 denote the number of elements in the top, middle, and bottom rows respectively. We can trivially check in linear time whether any row contains more than half of the objects; if so, we report that no solution is possible. We now assume there are at most $n/2$ objects per row. Suppose that P occupies m columns and assume that the input is not trivial. (See Observation 3 and Lemma 2.) Since we want to compress the configuration into a rectangle of width $\frac{n}{2}$, we perform at most $m-\frac{n}{2}$ (i.e., at most $\frac{n}{2}$) horizontal pushes. Notice that we will push vertically exactly once. As soon as that happens, by Lemma 2, it is trivial to check if a rectangle can be formed. Without loss of generality, we may assume that we are never pushing up, since we perform at most one vertical push. We need to determine the number of left pushes and the number of right pushes that should be performed before the single vertical push. By Lemma 1, we can perform those left and right pushes in any order. Thus we characterize each potential solution by its *push signature* (ℓ, r), where ℓ is the number of left pushes and r is the number of right pushes. Since $\ell + r \leq n$, we can simply try all possible push signatures with these constraints, of which there are a quadratic number. Checking if a push signature is feasible takes amortized constant time by maintaining, for each row, a pointer to the last empty space in the configuration, so this would yield a total time complexity of $O(n^2)$.

Consider a push signature $G = (\ell, r)$, after performing ℓ left pushes and r right pushes. G is *feasible* if there is no column containing 3 objects. G is *conforming* (resp. *sub-conforming*) if the number of columns where there are two objects in the top two rows is equal to (resp. less than) $(r_1 + r_2 - r_3)/2$.

Lemma 5 *A configuration of n objects can be compressed into a $2 \times \frac{n}{2}$ rectangle iff there exists a push signature (ℓ, r) that is both feasible and conforming.*

Now, we introduce the *feasibility diagram*. It is a 2-dimensional matrix where the number of left pushes and the number of right pushes are on the axes (see Figure 4). We mark the feasible cells.

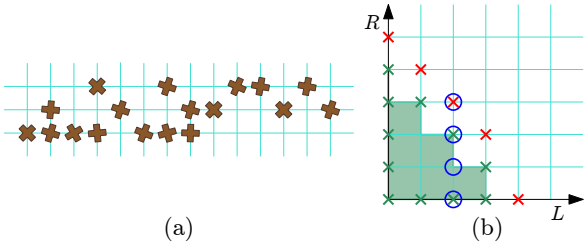


Figure 4: (a) A configuration (n_1=5, n_2=6,n_3=7) and (b) its feasibility diagram. The vertical axis is the number of right pushes, and the horizontal is the number of left pushes. The ×'s represent a computed feasibility push signature, where green is 'feasible' and red is 'not feasible'. The green shaded region shows the region of feasibility, which is all points below the staircase outlined by the computed feasible points. The blue ○'s mark a conforming push signature.

Lemma 6 *Any cells below and to the left of a feasible cell are also feasible.*

Proof. Let a feasible cell have coordinates (ℓ, r). If $(\ell-1, r)$ were not feasible, we could take that configuration and push left to produce a feasible configuration. This means that we would have decreased the

number of elements in a column, which is not possible by Observation 2. Therefore $(\ell-1, r)$ must also be feasible. The same applies to $(\ell, r-1)$. □

This property implies the Pareto-maximal cells form a staircase. We also mark the conforming cells, which form a x, y-monotone path. Finally, we check whether there is any feasible conforming cell.

Computing the feasible cells. A column of height 3 can only be created where an empty space is, and therefore, by keeping track of these spaces, we do not need to search the entire configuration every time we push. Using this fact, we can compute the feasibility of all push signatures that consist of only left or only right pushes in linear time, which will give us a maximum number of right and left pushes that are feasible. Given the maximum number R of right pushes that yield a feasible solution, we can then begin constructing the staircase; refer to Figure 4. We know that $(0, R)$ is feasible. If pushing left from the resulting configuration still yields a feasible solution, we note that $(1, R)$ is also feasible and continue. If pushing left is non-feasible, we decrement the number of right pushes we make, and see if pushing right that many times and then pushing left once yields a feasible solution. The crucial point is that we can test this in constant time, even though we cannot "unpush", by using the above observation. We continue in this way until we have drawn out a staircase in our diagram. By Lemma 6, we will characterize all feasible cells in the diagram in linear time.

Computing the conforming cells. After each push we add one element to one column for each row, which means that we can keep track of the number of columns we add elements to in constant time per push. At each push, we maintain the number of columns with two elements in the following way. If there is a column with one element in the middle row and one in the bottom row, then any push down will not change the number of elements in the resulting rows for this column. If there is a column with one element in the top row and one in the bottom row, then the same is true. However, if there is a column with one element in the top row and one element in the middle row, then when we push downward, the element from the middle row will be displaced into the bottom, and the element from the top row will be in the resulting top row. At each push, we maintain the number of columns that have a top-middle configuration, as well as the number of columns with the bottom-middle and top-bottom configurations. The number of top-middle configurations tells us how many objects will be added to the bottom row upon pushing, which will give us the point at which a push signature is conforming.

Any push-signature representing a solution will lie in the intersection of the conforming and feasible regions of the diagram. We compute both sets in linear time, and check if their intersection is empty.

Theorem 7 *Given any configuration P of n objects that occupy at most 3 rows, we can decide in $O(n)$ time whether P can be pushed into a $2 \times \frac{n}{2}$ rectangle.*

3.1 Rectangles of height k

The brute-force $O(n^2)$-time algorithm easily extends to rectangles of height $k > 2$. A solution now consists of exactly $k-2$ vertical pushes, and a number of left and right pushes between pairs of vertical pushes. We can encode this by a sequence of $k-2$ push signatures; since there are at most $n^{2(k-2)}$ such sequences.

Theorem 8 *If P has n objects occupying at most k rows, we can decide in $O(n^{2(k-2)})$ time whether P can be pushed into a $2 \times \frac{n}{2}$ rectangle.*

References

[1] A. Becker, E. Demaine, S. Fekete, and J. McLurkin. Particle computation: Controlling robot swarms with only global signals. In *ICRA*, pages 6751–6756, 2014.

[2] A. Becker, E. Demaine, S. Fekete, S. M. Shad, and R. Morris-Wright. Tilt: The video. designing worlds to control robot swarms with only global signals. In *Proc. 25th Multimedia Exp. Comp. Geom.*, 2015.

[3] E. D. Demaine, M. L. Demaine, M. Hoffmann, and J. O'Rourke. Pushing blocks is hard. *CGTA*, 26(1):21–36, 2003.

[4] A. Dhagat and J. O'Rourke. Motion planning amongst movable square blocks. In *Proc. 4th CCCG*, pages 188–191, 1992.

[5] D. El-Khechen, M. Dulieu, J. Iacono, and N. van Omme. Packing 2x2 unit squares into grid polygons is NP-complete. In *Proc. CCCG*, pages 17–19, 2009.

[6] P. Erdös and R. Graham. On packing squares with equal squares. *J. Comb. Theory*, 19:119–123, 1975.

[7] R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[8] E. Friedman. Packing unit squares in squares: A survey and new results. *The Electronic Journal of Combinatorics*, 2009.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] A. van Renssen and B. Speckmann. The 2x2 simple packing problem. In *Proc. 23rd CCCG*, 2011.

An Approximation Algorithm for the Two-Watchman Route in a Simple Polygon

Bengt J. Nilsson* Eli Packer†

Abstract

The *two-watchman route problem* is that of computing a pair of closed tours in an environment so that the two tours together see the whole environment and some length measure on the two tours is minimized. Two standard measures are: the minmax measure, where we want the tours where the longest of them has minimal length, and the minsum measure, where we want the tours for which the sum of their lengths is smallest. It is known that computing the minmax two-watchman route is NP-hard for simple rectilinear polygons and thus also for simple polygons. We exhibit a polynomial time 7.1416-factor approximation algorithm for computing the minmax two-watchman route in simple polygons.

1 Introduction

Some of the most intriguing problems in computational geometry concern visibility and motion planning in polygonal environments. A classical problem is that of computing a *shortest watchman route* in an environment, i.e., the shortest closed tour that sees the complete free-space of the environment. This problem has been shown NP-hard [5] and even $\Omega(\log n)$-inapproximable unless P=NP [7] for polygons with holes having a total of n segments.

Watchman route algorithms either compute a *fixed* watchman route which requires the tour to pass a given boundary point or they compute a *floating* watchman route, with no requirement to pass any specific point. Tan *et al.* [11] prove an $O(n^4)$ time algorithm based on dynamic programming for computing a shortest fixed watchman route through a given boundary point in a simple polygon. This is later improved to $O(n^3 \log n)$ time by Dror *et al.* [4]. Carlsson *et al.* [2] show how to generalize algorithms for the shortest fixed watchman route to compute a shortest floating watchman route in a simple polygon with a quadratic factor overhead. Tan [10] improves this to a linear factor overhead. Hence, the currently best algorithm for a shortest floating watchman route in a simple polygon uses $O(n^4 \log n)$ time.

*Department of Computer Science, Malmö University, SE-205 06 Malmö, Sweden. bengt.nilsson.TS@mah.se

†Proactive Location Intelligence Team, IBM, Haifa, Israel. ELIP@il.ibm.com

The problem of computing multiple watchman routes that together see the environment has received much less attention. Mitchell and Wynters [8] show that already computing the pair of tours that together see a simple rectilinear polygon is NP-hard, if we want to minimize the length of the longest of the two tours, the *minmax* measure. It is still an open problem whether it is possible to compute a pair of tours for which the sum of the lengths of the two tours is minimal, the *minsum* measure, in polynomial time. Packer [9] give some experimental results for multiple watchman routes in simple polygons. In the case when the watchmen are point sized, Belleville [1] shows an efficiently computable characterization of all simple polygons that are two-guardable with point guards.

We give a polynomial time 7.1416-factor approximation algorithm to compute a minmax pair of tours that together see a simple polygon.

2 Preliminaries

Let $\mathbf{P}$ be a simple polygon having n vertices and let $\partial\mathbf{P}$ denote the boundary of $\mathbf{P}$. We say that two points in $\mathbf{P}$ *see* each other, if the line segment connecting the points does not intersect the exterior of $\mathbf{P}$. For any connected object X inside $\mathbf{P}$, we denote by $\mathbf{VP}(X)$ the *weak visibility polygon* of X in $\mathbf{P}$, i.e., the set of points in $\mathbf{P}$ that see some point of X. $\mathbf{VP}(X)$ when X is a point, a segment, or a polygonal curve in $\mathbf{P}$ can be efficiently computed [6].

We define a *cut* to be a directed line segment in $\mathbf{P}$ with both end points on $\partial\mathbf{P}$ and having at least one interior point not on $\partial\mathbf{P}$. Hence, a polygon edge is not a cut. A cut separates $\mathbf{P}$ into two sub-polygons. If a cut is represented by the segment $[p, q]$ we say that the cut is directed from p to q and we call p the *start point* of the cut. For a cut c in $\mathbf{P}$, we let the *left polygon*, $\mathbf{L}(c)$, be the set of points in $\mathbf{P}$ locally to the left of c.

Assume a counterclockwise walk of $\partial\mathbf{P}$. Such a walk imposes a direction on each of the edges of $\mathbf{P}$ in the direction of the walk. Consider a reflex vertex of $\mathbf{P}$. The two edges incident to the vertex can each be extended inside $\mathbf{P}$ until the extensions reach a boundary point. These extended segments form cuts given the same direction as the edge they are collinear to. We call these cuts *extensions*.

A *guard set* is any set of points that together see

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

all of $\mathbf{P}$. Any guard set must have points intersecting $\mathbf{L}(e)$ for every extension e of $\mathbf{P}$, since otherwise the edge collinear to e will not be seen by the guard set. Chin and Ntafos [3] prove that this is indeed also a sufficient requirement when the guard set is connected, as it is for a shortest watchman route.

Let c be a cut. If a guard set $\mathcal{G}$ intersects $\mathbf{L}(c)$, we say that c is *covered* by $\mathcal{G}$. Furthermore, if $\mathcal{G}$ intersects the interior of $\mathbf{L}(c)$, then $\mathcal{G}$ *properly covers* c. If $\mathcal{G}$ properly covers c and intersects c, we say that $\mathcal{G}$ *crosses* c. Finally, if $\mathcal{G}$ covers c, but does not properly cover c, then $\mathcal{G}$ *reflects* on c.

For two cuts, c and c', we say that c *dominates* c', if $\mathbf{L}(c) \subseteq \mathbf{L}(c')$ An extension that is not dominated by any other extension is called *essential*. By the transitivity of the domination relation, if a guard set has points to the left of each essential extension, it also has points to the left of every extension [3].

All exact watchman route algorithms for simple polygons [2, 3, 4, 10, 11] compute closed tours that cover every essential extension. They can also be used with any set of cuts $\mathcal{C}$ to compute the shortest tour that covers each cut in $\mathcal{C}$, in polynomial time. We call such a tour the *shortest visiting tour* of the cuts in $\mathcal{C}$ inside $\mathbf{P}$ and denote it $SVT_{\mathcal{C}}$. For the case that $\mathcal{C}$ consists of the essential extensions of $\mathbf{P}$, the tour is a *shortest watchman route*, W_{S}.

We also make use of the fact that shortest paths in $\mathbf{P}$ between combinations of segments and points can be computed efficiently [6]. We denote the shortest path between two objects X and Y in $\mathbf{P}$ by $SP(X, Y)$.

Let X_1 and X_2 be two closed polygonal cycles contained in a simple polygon $\mathbf{P}$, such that any point in $\mathbf{P}$ sees some point on X_1 or X_2. We call such a pair $\mathcal{X} = (X_1, X_2)$, a *two-watchman route*. The length of a cycle X in $\mathbf{P}$ is denoted $||X||$ and we let $||\mathcal{X}||_{\mathsf{sum}} \stackrel{\text{def}}{=} ||X_1|| + ||X_2||$ be the *sum length* of $\mathcal{X}$ and $||\mathcal{X}||_{\mathsf{max}} \stackrel{\text{def}}{=} \max\{||X_1||, ||X_2||\}$ be the *max length* of $\mathcal{X}$.

Let $\mathcal{S} = (S_1, S_2)$ and $\mathcal{T} = (T_1, T_2)$ be two two-watchman routes such that $||\mathcal{S}||_{\mathsf{sum}} \leq ||\mathcal{X}||_{\mathsf{sum}}$ and $||\mathcal{T}||_{\mathsf{max}} \leq ||\mathcal{X}||_{\mathsf{max}}$ for any two-watchman route $\mathcal{X}$ in $\mathbf{P}$. We say that $\mathcal{S}$ is a *minsum* two-watchman route and $\mathcal{T}$ is a *minmax* two-watchman route. The following inequalities are immediate from the definitions,

$$||\mathcal{T}||_{\mathsf{max}} \leq ||\mathcal{S}||_{\mathsf{sum}} \leq 2||\mathcal{T}||_{\mathsf{max}}.$$

3 Approximating a Minimum Two-Watchman Route

Our algorithm is illustrated in pseudo-code in Figure 1 and we show that it approximates a minmax two-watchman route.

The algorithm begins by running Belleville's algorithm [1] to establish if the polygon is guardable by two point guards. If this is the case, it returns the two point guards computed by the algorithm. Otherwise, it computes, the set of essential extensions $\mathcal{E}$, a shortest watchman route W_{S} and initializes the solution to be two copies of W_{S}. The rest of this section is devoted to showing how to implement Step 5 of the algorithm.

Algorithm *Two-Watchman-Route*
Input: A simple polygon $\mathbf{P}$
Output: A two-watchman route $\mathcal{W}_{\mathsf{T}}$ that sees $\mathbf{P}$

1 Run Belleville's algorithm [1] to establish if the polygon is guardable by two point guards. If this is the case, return the two point guards computed by the algorithm
2 Compute the set of essential extensions $\mathcal{E}$ in $\mathbf{P}$
3 Compute a shortest watchman route $W_{\mathsf{S}} = SVT_{\mathcal{E}}$ in $\mathbf{P}$
4 Let $\mathcal{W}^*_{\mathsf{T}} := (W_{\mathsf{S}}, W_{\mathsf{S}})$
5 **for** every pair of extensions $e_1, e_2 \in \mathcal{E}$, $e_1 \neq e_2$ **do**
5.1 Compute the V-structure $\mathcal{V}_{e_1,e_2}$ and establish its bases q_1 and q_2
5.2 Let $\mathcal{F}_1 := \emptyset$ and $\mathcal{F}_2 := \emptyset$
5.3 **for** every boundary edge $b = [v, v']$ **do**
Compute the minimum tentacle pair $\mathcal{Z}^{\min}_{q_1,q_2}(b) = \mathcal{Z}^{r}_{q_1,q_2}(b)$ giving r on b
if b is a double edge ($r \neq v, v'$) **then**
Let c_1 and c_2 be the cuts through r and the end points of $\mathcal{Z}^{\min}_{q_1}(b)$ and $\mathcal{Z}^{\min}_{q_2}(b)$
Add c_1 to $\mathcal{F}_1$ and c_2 to $\mathcal{F}_2$
else /* b is a single edge ($r = v$ or $r = v'$) */
if $\mathcal{Z}^{\min}_{q_1}(b)$ sees b **then**
Let c and c' be the cuts through v, v' and the end point of $\mathcal{Z}^{\min}_{q_1}(b)$
Add c and c' to $\mathcal{F}_1$
else /* $\mathcal{Z}^{\min}_{q_2}(b)$ sees b */
Let c and c' be the cuts through v, v' and the end point of $\mathcal{Z}^{\min}_{q_2}(b)$
Add c and c' to $\mathcal{F}_2$
5.4 Compute the two tours $\mathcal{W}_{\mathsf{T}} = (SVT_{\mathcal{F}_1}, SVT_{\mathcal{F}_2})$
5.5 **if** $||\mathcal{W}_{\mathsf{T}}||_{\mathsf{max}} < ||\mathcal{W}^*_{\mathsf{T}}||_{\mathsf{max}}$ **then** $\mathcal{W}^*_{\mathsf{T}} := \mathcal{W}_{\mathsf{T}}$
6 **return** $\mathcal{W}^*_{\mathsf{T}}$
End *Two-Watchman-Route*

Figure 1: The Two-Watchman-Route algorithm.

We claim the following lemma without proof.

Lemma 1 *If two tours in $\mathbf{P}$ see all of $\partial\mathbf{P}$, then they see all of $\mathbf{P}$.*

The lemma implies that it is sufficient to construct two tours that see the whole boundary of $\mathbf{P}$ to guarantee that all of $\mathbf{P}$ is guarded.

There is a partitioning of the extensions in $\mathcal{E}$ into nonempty subsets $\mathcal{E}_1$ and $\mathcal{E}_2$, such that each tour T_i of a minmax two-watchman route covers the extensions in $\mathcal{E}_i$, $i \in \{1, 2\}$. We even have a stronger claim.

Lemma 2 *Each tour T_i in a minmax two-watchman route $\mathcal{T} = (T_1, T_2)$ intersects some extension in $\mathcal{E}_i$.*

Consider two tours X_1 and X_2 and a polygon boundary edge b. We claim the following lemma.

Lemma 3 *For any two tours X_1 and X_2 and a polygon boundary edge b, the sets $\mathbf{VP}(X_i) \cap b$ and $\mathbf{VP}(X_1) \cap \mathbf{VP}(X_2) \cap b$ are each connected.*

For a point q (or an extension e) in $\mathbf{P}$ and a (possibly point sized) subsegment s_b of boundary edge b, we call the shortest path from q (or e) to some point

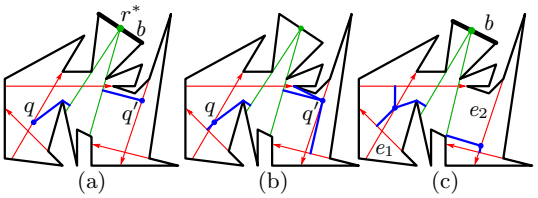


Figure 2: (a) A tentacle pair $\mathcal{Z}^{\min}_{q,q'}(b)$, (b) a jellyfish pair $\mathcal{J}_{q,q'}$, (c) a minimum jellyfish pair $\mathcal{J}^{\min}_{e_1,e_2}$.

in $\mathbf{P}$ that sees all points of s_b a *tentacle* from q (or e) to s_b, denoted $Z_q(s_b)$ (or $Z_e(s_b)$).

For a boundary segment $b = [v, v']$ and a point r on b, we let $b(r)$ be the subsegment $[v, r]$ and $\bar{b}(r)$ be the subsegment $[r, v']$. For two points q and q' and a point r on b, the *tentacle pair* that sees b is the shorter of the pairs $(Z_q(b(r)), Z_{q'}(\bar{b}(r)))$ and $(Z_q(\bar{b}(r)), Z_{q'}(b(r)))$. We denote this pair $\mathcal{Z}^r_{q,q'}(b)$ and define its length to be the length of the longer of the two tentacles in the pair.

For some point r^* on b, it holds that $||\mathcal{Z}^{r^*}_{q,q'}(b)|| \leq \min_{r\in b}\{||\mathcal{Z}^r_{q,q'}(b)||\}$. If r^* is one of the end points of b, one of the tentacles in the tentacle pair degenerates into a single point q or q'. We denote this minimum tentacle pair by $\mathcal{Z}^{\min}_{q,q'}(b)$. The two tentacles attached to q and q' are denoted $\mathcal{Z}^{\min}_{q}(b)$ and $\mathcal{Z}^{\min}_{q'}(b)$ respectively; see Figure 2(a); and we have that

$$||\mathcal{Z}^{\min}_{u_1,u_2}(b)|| \leq ||(T_1, T_2)||_{\max}/2, \tag{1}$$

where u_1 and u_2 are intersection points of T_1 and T_2 with e_1 and e_2 respectively. The inequality holds since T_1 and T_2 together see b.

For two points q and q' in $\mathbf{P}$, we call $\mathcal{J}_{q,q'} = \{\mathcal{Z}^{\min}_{q,q'}(b) \mid b \in \partial\mathbf{P}\}$ the *jellyfish pair* with origins q and q'; see Figure 2(b). We define the length of a jellyfish pair to be the length of its longest tentacle.

We define the *bases* along segments s and s' to be a pair of points $(q_*, q'_*) = \arg\min_{q\in s, q'\in s'}\{||\mathcal{J}_{q,q'}||\}$, i.e., two points q_* on s and q'_* on s' where $||\mathcal{J}_{q_*,q'_*}||$ is minimal. We denote the jellyfish pair $\mathcal{J}_{q_*,q'_*}$ by $\mathcal{J}^{\min}_{s,s'}$. From this definition and (1), we have

$$||\mathcal{J}^{\min}_{e_1,e_2}|| \leq ||\mathcal{J}_{u_1,u_2}|| \leq ||(T_1, T_2)||_{\max}/2. \tag{2}$$

We can select two longest tentacle pairs of $\mathcal{J}^{\min}_{e_1,e_2}$, at least one pair of which attains the length $||\mathcal{J}^{\min}_{e_1,e_2}||$. The two tentacle pairs have two bases q_1 on e_1 and q_2 on e_2, one pair is the shortest tentacle pair $\mathcal{Z}^{\min}_{q_1,q_2}(b)$, the other is the shortest tentacle pair $\mathcal{Z}^{\min}_{q_1,q_2}(b')$, for boundary edges b and b'. We call the two tentacle pairs that attain the maximum length a *V-structure* on e_1 and e_2, and denote it $\mathcal{V}_{e_1,e_2}$. The length of $\mathcal{V}_{e_1,e_2}$ is the length of its longest tentacle. From this definition and (2) we have

$$||\mathcal{V}_{e_1,e_2}|| = ||\mathcal{J}^{\min}_{e_1,e_2}|| \leq ||(T_1, T_2)||_{\max}/2. \tag{3}$$

The algorithm needs to find the two bases q_1 on e_1 and q_2 on e_2. Therefore, the algorithm must determine the two boundary edges b and b', and the two points r and r' on b and b' for which the maximum length of the V-structure is attained. Since we do not know which pair of boundary edges produce the V-structure that attains the length of $\mathcal{J}^{\min}_{e_1,e_2}$, we try all possible pairs of boundary edges $b_i = [v_i, v'_i]$ and $b_j = [v_j, v'_j]$, $1 \leq i \leq j \leq n$ in Step 5.1 of the algorithm. We allow $i = j$ to take care of the case when the longest tentacle in $\mathcal{J}^{\min}_{e_1,e_2}$ is unique.

In Step 5.1, we begin by computing $Z_{e_1}(b_i)$ and $Z_{e_1}(b_j)$ as well as the two pairs $Z_{e_2}(v_i)$, $Z_{e_2}(v'_i)$ and $Z_{e_2}(v_j)$, $Z_{e_2}(v'_j)$. Assume that $Z_{e_2}(v_i)$ and $Z_{e_2}(v_j)$ are the shorter of the two tentacles in each pair.

We obtain the two points q and q' on the extensions e_1 and e_2 that minimize $\max\{||Z_q(b_i)||, ||Z_q(b_j)||\}$ and $\max\{||Z_{q'}(v_i)||, ||Z_{q'}(v_j)||\}$. We let two points r_i on b_i and r_j on b_j slide independently, r_i from v_i to v'_i and r_j from v_j to v'_j. We can express the position on e_1 of q and on e_2 of q' as functions of r_i and r_j, and hence also the expressions $\max\{||Z_q(b_i(r_i))||, ||Z_q(b_j(r_j))||\}$ and $\max\{||Z_{q'}(\bar{b_i}(r_i))||, ||Z_{q'}(\bar{b_j}(r_j))||\}$.

The difference between these two expressions is a multivariate function $\mathcal{D}_{ij}(r_i, r_j)$ on r_i and r_j that locally only depends on the contact points of the supporting segments for r_i and r_j and the corresponding paths $Z_q(b_i(r_i))$, $Z_q(b_j(r_j))$, $Z_{q'}(\bar{b_i}(r_i))$, and $Z_{q'}(\bar{b_j}(r_j))$, a total of at most eight polygon vertices. We compute the values of r_i and r_j that produce the minimum absolute value $|\mathcal{D}_{ij}(r_i, r_j)|$ in all intervals for r_i and r_j where the contact points do not change.[1] As r_i moves from v_i to v'_i, the supporting lines for r_i can change at most $O(n)$ times and the same holds for r_j so in at most $O(n^2)$ time the minimum can be obtained. We maintain the pair of bases q and q' for which the corresponding V-structure $\mathcal{V}_{e_1,e_2}$ has maximum length. We denote these points by q_1 and q_2.

Given q_1 and q_2, we compute, in Step 5.3, the minimum tentacle pairs $\mathcal{Z}^{\min}_{q_1,q_2}(b) = \mathcal{Z}^r_{q_1,q_2}(b)$ for every boundary edge $b = [v, v']$, giving us the minimum jellyfish pair on e_1 and e_2, $\mathcal{J}^{\min}_{e_1,e_2}$. If the expression $\max\{||\mathcal{Z}^{\min}_{q_1}(b)||, ||\mathcal{Z}^{\min}_{q_2}(b)||\}$ is minimized for $r = v$ or $r = v'$, then b is a *single edge*, otherwise it is a *double edge*. If b is a double edge, the point r and the endpoint of $\mathcal{Z}^{\min}_{q_1}(b)$ different from q_1 defines a cut in $\mathbf{P}$ that passes through the two points. The direction of the cut is such that q_1 does not lie to the left of the cut and is added to the set $\mathcal{F}_1$. We also construct the cut through r and the endpoint of $\mathcal{Z}^{\min}_{q_2}(b)$ different from q_2. This cut is directed so that q_2 does not lie to the left of the cut and is added to the set $\mathcal{F}_2$. The green segments in Figure 2(c) are the two cuts for boundary edge b.

Similarly, if b is a single edge, it is seen by one of $\mathcal{Z}^{\min}_{q_1}(b)$ or $\mathcal{Z}^{\min}_{q_2}(b)$. If it is seen by $\mathcal{Z}^{\min}_{q_1}(b)$, the endpoints v and v' of b together with the endpoint of

[1] We assume a real RAM computational model that allows us to compute arbitrary algebraic functions and roots of algebraic functions.

$\mathcal{Z}^{\min}_{q_1}(b)$ different from q_1 define two cuts. The direction of the cuts are such that q_1 does not lie to the left of them and they are added to the set $\mathcal{F}_1$. If b is seen by $\mathcal{Z}^{\min}_{q_2}(b)$, we construct two cuts in the same way and add these to $\mathcal{F}_2$.

To finalize, we let $W_1 = SVT_{\mathcal{F}_1}$ and $W_2 = SVT_{\mathcal{F}_2}$, two shortest visiting tours of the cut sets $\mathcal{F}_1$ and $\mathcal{F}_2$, and return the pair (W_1, W_2) as our two-watchman route.

Lemma 4 *The tours (W_1, W_2) obtained by algorithm Two-Watchman-Route form a two-watchman route and $||(W_1, W_2)||_{\max} \leq (\pi + 4)||(T_1, T_2)||_{\max}$.*

Proof. (Sketch) It follows from Lemma 1 and the fact that the two tours together see every boundary edge that they form a two-watchman route.

The algorithm computes the minimum jellyfish pair $\mathcal{J}^{\min}_{e_1,e_2}$ in the loop of Step 5.3. By trying all pairs of extensions in Step 5, the algorithm must necessarily consider a pair intersected by the tours T_1 and T_2; see Lemma 2. Consider the tentacles attached to the base q_1 on e_1. If we follow the shortest path from each tentacle endpoint not on q_1 to the next, cyclically around q_1, we obtain a tour U_1 that visits every cut in the set $\mathcal{F}_1$. Every tentacle has length at most $R = ||(T_1, T_2)||_{\max}/2$ by (2), hence U_1 is inscribed in a circle of radius R. Since $||W_1|| \leq ||U_1||$, the convex chains of W_1 together have length $\leq 2\pi R$.

If T_1 intersects T_2, then $||W_{\mathsf{s}}|| \leq ||T_1|| + ||T_2|| \leq 2||(T_1, T_2)||_{\max}$ proving the lemma since $||(W_1, W_2)||_{\max} \leq ||W_{\mathsf{s}}||$.

If T_1 does not intersect T_2 and W_1 has at most four reflex chains, then $||W_1|| \leq 2\pi R + 8R \leq (\pi + 4) \cdot ||(T_1, T_2)||_{\max}$.

If T_1 does not intersect T_2, W_1 has at least five reflex chains and W_1 does not intersect T_2, then use the segments of W_1 to cut $\mathbf{P}$, thus partitioning $\mathbf{P}$ into separate components. Let $\mathbf{Q}$ be the component containing T_2. The convex chain W_{c} of W_1 bounding $\mathbf{Q}$ has length $\leq 2\pi R$. The two reflex chains of W_1 adjacent to W_{c} have length $\leq 4R$ and the remainder of W_1 follows the same path as T_1, giving us $||W_1|| \leq ||T_1|| + 4R + 2\pi R \leq (\pi + 3)||(T_1, T_2)||_{\max}$.

Finally, if W_1 intersects T_2, then use T_2 to cut $\mathbf{P}$, partitioning it into components. Let $\mathbf{Q}'$ be the component containing T_1. The intersection $W'_{\mathsf{c}} = W_1 \cap \mathbf{Q}'$ follows the same path as T_1, the two reflex chains of W_1 adjacent to W'_{c} have length $\leq 4R$ and the remaining reflex chains of $W''_{\mathsf{c}} = W_1 \cap (\mathbf{P} \setminus \mathbf{Q}')$ follow T_2. The convex chains of W''_{c} have total length $\leq 2\pi R$ so we have $||W_1|| \leq ||T_1|| + 4R + 2\pi R + ||T_2|| \leq (\pi + 4)||(T_1, T_2)||_{\max}$.

We bound W_2 similarly, proving the lemma. □

The analysis of the algorithm is straightforward. The for-loop in Step 5 considers $O(n^2)$ pairs of extensions. Computing $\mathcal{V}_{e_1,e_2}$ takes $O(n^4)$ time by going through all pairs of boundary edges. The work in the remaining steps of the outermost for-loop is dominated by the cost of computing the shortest visiting tours in Step 5.4 taking $O(n^4 \log n)$ time. Hence, the total time complexity for the algorithm is $O(n^6 \log n)$.

Theorem 5 *The Two-Watchman-Route algorithm computes a 7.1416-approximation of the minmax two-watchman route in $O(n^6 \log n)$ time.*

4 Conclusions

Our algorithm relies heavily on the fact that for two tours it is sufficient to guarantee that the boundary is seen to ensure that the complete polygon is seen. This does not hold for three or more tours. It is therefore very possible that the problem is inapproximable for three watchmen.

Establishing the complexity for the minsum two-watchman route is still open although our algorithm provides a polynomial 14.2832-approximation.

The authors would like to thank Paweł Żyliński for fruitful discussions.

References

[1] P. Belleville. Two-guarding simple polygons. Master's thesis, School of Computer Science, McGill University, September 1991.

[2] S. Carlsson, H. Jonsson, and B.J. Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete and Computational Geometry*, 22:377–402, 1999.

[3] W. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.

[4] 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. ACM, 2003.

[5] A. Dumitrescu and C.S. Tóth. Watchman tours for polygons with holes. *Computational Geometry: Theroy and Applications*, 45(7):326–333, 2012.

[6] J.E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*, 2nd ed. CRC Press, 2004.

[7] J.S.B. Mitchell. Approximating watchman routes. In *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms, SODA'13*, pages 844–855, 2013.

[8] J.S.B. Mitchell and E.L. Wynters. Watchman routes for multiple guards. In *Proc. 3rd Canadian Conference on Computational Geometry*, pages 126–129, 1991.

[9] E. Packer. Computing multiple watchman routes. In *Proc. 7th International Workshop, WEA 2008*, pages 114–128. Springer Verlag, Lecture Notes in Computer Science 5038, 2008.

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

[11] 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.

A PTAS for Euclidean Maximum Scatter TSP

László Kozma* Tobias Mömke*†

Abstract. We study the problem of finding a tour of n points in $\mathbb{R}^d$ in which *every edge is long*. More precisely, we wish to find a tour that maximizes the length of the shortest edge in the tour. The problem is known as Maximum Scatter TSP, and it was introduced by Arkin et al. (SODA 1997), motivated by applications in manufacturing and medical imaging. Arkin et al. gave a 2-approximation for the metric version of the problem and showed that this is the best possible ratio achievable in polynomial time (assuming P ≠ NP). They raised the question of whether one can obtain a better approximation ratio in the planar Euclidean case. We answer this question in the affirmative in a more general setting, by giving a polynomial-time approximation scheme (PTAS) for Maximum Scatter TSP in an arbitrary fixed-dimensional Euclidean space.

1 Introduction

Let $P = \{p_1, \ldots, p_n\}$ be a set of points in $\mathbb{R}^d$. A tour T of P is a sequence $T = (p_{i_1}, \ldots, p_{i_n})$, where $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$. The *scatter* of tour T is the minimum distance between neighboring points of T, i. e., $\min\{d(p_{i_1}, p_{i_2}), \ldots, d(p_{i_{n-1}}, p_{i_n}), d(p_{i_n}, p_{i_1})\}$. The Maximum Scatter Travelling Salesman Problem (MSTSP) asks for a tour of P with maximum scatter. We study this problem in the *geometric* setting where the distance function d is the Euclidean distance between points.

Arkin et al. [1] initiated the study of MSTSP in 1997, motivated by problems in manufacturing (riveting) and medical imaging. They gave a simple 2-approximation algorithm for the more general metric problem (where distances are only required to satisfy the triangle inequality). They also showed that for the metric variant, the approximation ratio of 2 is optimal (assuming P ≠ NP). It was left open whether a better approximation ratio can be obtained in polynomial time if the problem has more geometric structure (e. g., if distances are Euclidean). Arkin et al. raise this question for the planar case (see also [6] and [13, p. 681]).

It is natural to expect that geometric structure should lead to stronger approximation-guarantees. The same phenomenon has been observed for the standard TSP problem: for metric TSP the best known approximation ratio is 1.5 (Christofides [5]), with a current lower bound of $\frac{123}{122}$ (Karpinski et al. [9]), whereas for Euclidean TSP Arora [2] and Mitchell [11] independently obtained polynomial-time approximation schemes (PTAS). Similarly, the Euclidean MaxTSP problem (where the goal is to maximize the *total* length of the tour) admits a PTAS (Barvinok [3]), but the metric version is currently known to admit only a $\frac{7}{8}$-approximation (Kowalik and Mucha [10]).

In this paper we answer the open question about planar MSTSP, by giving a polynomial-time $(1-\epsilon)$-approximation, for arbitrary fixed $\epsilon > 0$. In fact, we present a PTAS for MSTSP in arbitrary fixed-dimensional Euclidean spaces. Since MSTSP is known to be strongly NP-complete in dimensions 3 and above [7], our result settles the classical complexity status of the problem in these dimensions. We show the following result.

Theorem 1 *Let P be a set of n points in $\mathbb{R}^d$. A tour of P whose scatter is at least a $(1-\epsilon)$ factor of the MSTSP optimum can be found in time $O\left(n^{(100d/\epsilon^2)^d}\right)$.*

Further related work. TSP is one of the cornerstones of combinatorial optimization and several variants have been considered in the literature (we refer to [8] for a survey). Minimizing variants are more common, but there exist natural settings in which tours with long edges are desirable. This is the case in certain manufacturing operations where nearby elements in a sequence are required to be geometrically well-separated in order to avoid interferences [1].

MSTSP (a.k.a. max min TSP) appears similar to Bottleneck TSP (a.k.a. min max TSP), a problem known to be NP-complete already in the planar Euclidean case [8]. For metric Bottleneck TSP, 2-approximation is the best possible [12], and we are not aware of stronger approximation-results for geometric variants. Despite the similarity between MSTSP and Bottleneck TSP, it is unclear whether any techniques can be transferred from one problem to the other.

Open question. Our current work does not address the complexity status of solving MSTSP *exactly* in the planar Euclidean case. It remains open whether this problem is NP-hard (the situation is the same for MaxTSP). We note that this question has a natural equivalent formulation: is checking for existence of a Hamiltonian cycle NP-complete in complements of unit disk graphs?

*Department of Computer Science, Saarland University, kozma@cs.uni-saarland.de, moemke@cs.uni-saarland.de

†Partially funded by Deutsche Forschungsgemeinschaft grant BL511/10-1 and MO 2889/1-1.

2 The PTAS (Proof of Theorem 1)

Consider a set P of n points in $\mathbb{R}^d$, a threshold value ℓ, and a precision parameter $\epsilon > 0$. Given these inputs, a PTAS for the MSTSP problem is an algorithm with running time polynomial in n, required to return "yes" if a tour of P exists with scatter (i. e., shortest length) at least ℓ. The algorithm is required to return "no" if there is no tour of P with scatter at least $\ell(1-\epsilon)$, and is otherwise allowed to return "yes" or "no" arbitrarily.

Such a PTAS is an approximation algorithm for the *decision version* of the MSTSP problem. Observe that the optimum value of the MSTSP problem can only take one of $\binom{n}{2}$ possible values (the distances between points in P). Thus, a binary search over the possible values turns a PTAS of the above kind into a PTAS for the optimization problem. In the following, we focus on the decision problem. Before proceeding to the algorithm, we present some structural observations upon which the algorithm relies.

Given a point set $P \in \mathbb{R}^d$, let G_P be a graph with vertex set $V(G_P) = P$ and edge set $E(G_P) = \{\{x, y\} \mid x, y \in P \ \wedge \ d(x, y) \geq \ell\}$. In words, G_P contains all edges with length at least ℓ. The MSTSP decision problem asks whether G_P contains a Hamiltonian cycle. The following result is well-known (see e. g., [4]), and is also used by Arkin et al.

Lemma 2 (Dirac's theorem) *A graph G with n vertices has a Hamiltonian cycle if the degree of every vertex in G is at least $\frac{n}{2}$. Furthermore, in such a case, a Hamiltonian cycle can be found in $O(n^2)$ time.*

Observe that if the condition of Lemma 2 holds for G_P, then we are done. If that is not the case, then there is a vertex in G_P, whose degree is less than $\frac{n}{2}$. In other words, there is a point $p \in P$, such that $|B_p \cap P| > \frac{n}{2}$, where B_p is the open ball of radius ℓ with center p. Let us fix p to be such a point, and let B'_p be the open ball of radius 2ℓ with center p. We show that the optimal solution can be assumed to have a certain structure in relation to B_p and B'_p.

Lemma 3 *Suppose a tour T of P with scatter at least ℓ exists. Then there exists a tour T' of P with scatter at least ℓ, such that for every pair $x, y \in P$ of neighboring points in T', at least one of x and y is contained in B'_p.*

Proof. Suppose this is not the case. Since B_p contains more than half of the points in P, it must contain at least one edge of T entirely. Let $\{z, t\}$ be such an edge. Since both x and y are outside of B'_p we have $d(x, t), d(x, z), d(y, t), d(y, z) \geq \ell$. Thus, we can replace the edges $\{x, y\}$ and $\{z, t\}$ in T, with either $\{x, z\}$ and $\{y, t\}$, or $\{x, t\}$ and $\{y, z\}$, depending on the ordering of the points in T. We obtain another tour with scatter at least ℓ, that no longer contains the edge $\{x, y\}$. We proceed in the same way until we have removed all edges with both endpoints outside of B'_p. See Fig. 1 for an illustration. □

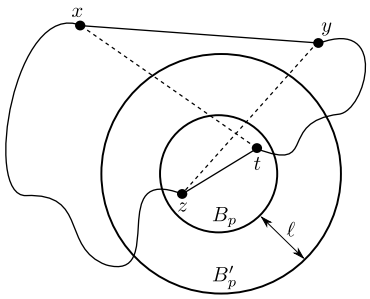


Fig. 1: Illustration of Lemma 3. The dashed edges can replace {x,y} and {z,t} in the optimal tour.

The next ingredient of the algorithm is a coarsening of the input, by rounding points in P to points of a grid. Let $\mathbb{G}_\delta$ be a δ-scaling of the d-dimensional unit grid, i. e., $\mathbb{G}_\delta = \{\delta(n_1, \ldots, n_d) \mid n_1, \ldots, n_d \in \mathbb{Z}\}$, for an arbitrary $\delta > 0$. Let f_δ (or simply f) be the mapping from $\mathbb{R}^d$ to $\mathbb{G}_\delta$ that maps each point to its nearest grid point (breaking ties arbitrarily). The following properties result from basic geometric considerations.

Lemma 4 *With f and δ as defined earlier, we have:*
(i) $d(x, y) \geq d(f(x), y) - \delta\sqrt{d}/2$ for all $x, y \in \mathbb{R}^d$,
(ii) $|B \cap \mathbb{G}_\delta| \leq (2\ell/\delta + 1)^d$ for every open ball B of radius ℓ.

Observe that f maps the graph G_P to a multigraph H_P defined as follows. Let $V(H_P) = \{v \mid v = f(x), x \in P\}$, i. e., the set of grid points with at least one mapped point of P, and let $E(H_P) = \{\{u, v\} \mid u = f(x), v = f(y), \{x, y\} \in E(G_P)\}$, i. e., the pairs of grid points to which edges of G_P are mapped. We also maintain multiplicities on edges of H_P, i. e., we keep track of how many edges of G_P are mapped to each edge of H_P.

A tour T of P (i. e., a Hamiltonian cycle of G_P) is mapped by f to an *Eulerian tour* of H_P. It is not hard to see that given this Eulerian tour of H_P, a tour of P can be recovered (by replacing multiple occurrences of a grid point with the points in P that are mapped to it). Moreover, the scatter of the recovered tour is not far from that of T (by Lemma 4(i)). However, the edges of H_P are not available to us, and thus it seems prohibitively expensive to guess a correct Eulerian tour on the vertices of H_P (there may be $\Omega(n)$ vertices). The key insight of the algorithm is that it is sufficient to consider the portion of H_P that falls inside B'_p.

At a high level, the strategy to obtain an approximation is the following (see Fig. 2 for an illustration). Assuming that the optimal tour has the property from Lemma 3, it consists of edges inside B'_p, and "hops" of two consecutive edges, connecting a point outside B'_p with two points inside B'_p. We replace such hops with virtual edges, both of whose endpoints are in B'_p. The

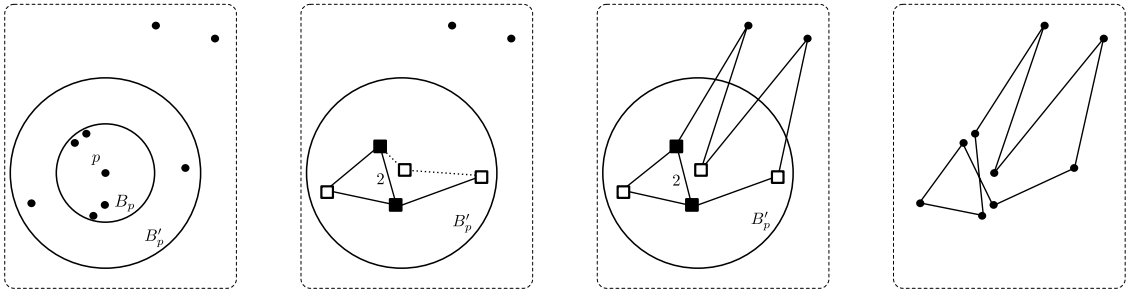


Fig. 2: Illustration of the algorithm. (i) Input point set with open balls B_p and B'_p with center p and radii ℓ and 2ℓ respectively. (ii) Points inside B'_p mapped to grid points (shown as squares), and "guessed" edges. Filled squares indicate grid points to which more than one point is mapped. Dotted lines indicate virtual edges, and the number indicates the multiplicity of an edge (omitted if 1). (iii) Virtual edges matched to points outside of B'_p and extended to hops, resulting in a multigraph. (iv) An Eulerian tour of the multigraph, expanded to a tour on the initial point set.

resulting tour is entirely in B'_p, and we can "guess" its image under f. This is now feasible, since the number of grid points involved is bounded by Lemma 4(ii). We also guess the multiplicities of all edges, i. e., how many original edges have been mapped to each edge, and how many edges are virtual.

We then disambiguate the virtual edges, i. e., we find a suitable midpoint outside of B'_p for each hop. This is achieved by solving a perfect matching problem. We obtain a multigraph on which we find an Eulerian tour. Finally, from the Eulerian tour we recover a tour of P. The distortion in distances due to rounding (i. e., the approximation ratio) is controlled by the choice of the grid resolution δ.

We only focus on answering whether a tour with the required scatter value exists. It will be clear that *constructing* such a tour can be achieved with minor changes to the algorithm. More details follow.

Algorithm. Input: a set P of n points in $\mathbb{R}^d$, a threshold ℓ, and a precision parameter $\epsilon > 0$.

1. Set $\delta = \epsilon\ell/(2\sqrt{d})$, and let $\ell' = \ell(1-\epsilon/2)$.
2. Find $p \in P$ such that $|B_p \cap P| > \frac{n}{2}$, where B_p and B'_p are the open balls with center p of radius ℓ and 2ℓ. If no such p exists, output Yes.
3. Let $f\colon P \to \mathbb{G}_\delta$ map points to their nearest grid point. Compute the set $C = \{f(x) \mid x \in (P \cap B'_p)\}$, and for each $v \in C$, compute the sets $f^{-1}(v) = \{x \mid f(x) = v\}$.
4. Let $m, v : \binom{C}{2} \to \mathbb{N}$. For all $\{u,v\} \subseteq C$, guess $m(\{u,v\})$ and $v(\{u,v\})$, such that
   (i) $m(\{u,v\}) = 0$ if $d(u,v) < \ell'$, and
   (ii) for all $v \in C$:
   $$\sum_{u \in C \setminus \{v\}} \big(m(\{u,v\}) + v(\{u,v\})\big) = 2|f^{-1}(v)|.$$
5. Construct a bipartite graph B as follows:
   - for each $\{u,v\} \subseteq C$, add $v(\{u,v\})$ vertices labeled $\{u,v\}$ to left vertex set $L(B)$.
   - for each $x \in P \setminus B'_p$ add a vertex labeled x to the right vertex set $R(B)$.
   - add an edge $(\{u,v\}, x)$ between $\{u,v\} \in L(B)$ and $x \in R(B)$ to $E(B)$ iff $d(u,x), d(v,x) \geq \ell'$.
6. Find a perfect matching M of B; if there is none, output No.
7. Construct a multigraph H as follows:
   - let $V(H) = C \cup (P \setminus B'_p)$.
   - for all $\{u,v\} \subseteq C$ add $m(\{u,v\})$ copies of the edge $\{u,v\}$ to $E(H)$.
   - for all $(\{u,v\}, x) \in M$ add the edges $\{u,x\}$ and $\{v,x\}$ to $E(H)$.
8. Find an Eulerian tour Q of H; if there is none, output No.
9. Transform Q into a tour T of P, by replacing multiple occurrences of every point $v \in C$, with the points in $f^{-1}(v)$ in arbitrary order.
10. Output Yes.

Note. The "guessing" in step 4 should be thought of as a loop over all possible values of m and v satisfying the requirements. The overall output is No if the output is No for all possible values of step 4.

Correctness. We prove two claims which together imply that the algorithm is a PTAS for MSTSP: (1) if the algorithm outputs Yes, then there is a tour of P with scatter at least $\ell(1-\epsilon)$, and (2) if there is a tour of P with scatter at least ℓ, then the algorithm outputs Yes.

(1) If we obtain Yes in step 2, then we have a tour with scatter at least ℓ by Lemma 2. Suppose that the algorithm returns Yes in step 10. This means that steps 5–9 were successful with the values of m and v chosen in step 4, and T is a tour of P. Consider an arbitrary edge $\{x,y\}$ of T. By step 9, there is a corresponding edge $\{u,v\}$ in the Eulerian tour Q of H. By construction of H in step 7, either (a) $u, v \in C$, or (b) $(\{u,w\}, v) \in M$ or $(\{v,w\}, u) \in M$, for some grid point $w \in C$.

In case (a) by condition (i) of step 4, we have $d(u,v) \geq \ell'$. Since $\{u,v\} = \{f(x), f(y)\}$, from Lemma 4(i) we obtain $d(x,y) \geq \ell' - \delta\sqrt{d} = \ell(1-\epsilon)$.

In case (b) by the construction of B in step 5, we have $d(u,v) \geq \ell'$. Since $\{u,v\}$ equals either $\{f(x),y\}$ or $\{x,f(y)\}$, from Lemma 4(i) we obtain $d(x,y) \geq \ell' - \delta\sqrt{d}/2 \geq \ell(1-\epsilon)$.

(2) Assume now that a tour T of P with scatter at least ℓ exists, and that the solution is not trivially found in step 1. Assume also w. l. o. g. that T has the special structure described in Lemma 3, i. e., it consists of hops and of edges entirely inside B'_p. Consider an edge $\{x,y\}$ of T, such that $x,y \in B'_p$. Then, after step 3, $f(x), f(y) \in C$ holds, and we say that $\{x,y\}$ *maps* to $\{f(x),f(y)\}$. Consider now a hop of T, i. e., two consecutive edges $\{x,w\}$ and $\{w,y\}$, such that $x,y \in B'_p$ and $w \in P \setminus B'_p$. Then, after step 3, $f(x), f(y) \in C$ holds, and we say that the hop $\{x,w,y\}$ *virtually maps* to $\{f(x),f(y)\}$.

Consider now the values m and v guessed in step 4, and let $m^*(\{u,v\})$ be the number of edges in T that map to $\{u,v\}$, and let $v^*(\{u,v\})$ be the number of hops in T that virtually map to $\{u,v\}$. Since every point in T has degree 2, it follows that the number of edges and hops mapped to an edge incident to some $u \in C$ is twice the number of points in P mapped to u. Furthermore, for all edges $\{x,y\} \subseteq B'_p$ of T, we have $d(f(x),f(y)) \geq \ell - \delta\sqrt{d} = \ell'$ (by Lemma 4(i)). Therefore, guessing the correct values $m = m^*$ and $v = v^*$ is consistent with the conditions in step 4.

Let $\{x_1,w_1,y_1\},\ldots,\{x_k,w_k,y_k\}$ denote all the hops in T, where $w_i \in P \setminus B'_p$, for all i. Let $u_i = f(x_i)$, and $v_i = f(y_i)$, and let us call $M(T) = \big\{\big(\{u_i,v_i\},w_i\big) \mid i = 1,\ldots,k\big\}$ the *hop-matching* of T. Observe that $M(T)$ is a valid perfect matching for the graph B constructed in step 5, therefore, step 6 will succeed. We cannot, however, guarantee that $M(T)$ will be revovered in step 6. Observe that any other perfect matching M of B corresponds to a shuffling of the points w_i in B, and thus it is a hop-matching of a tour T' in which the points w_i have been correspondingly shuffled. T' differs from T only in its hops, and by construction of B in step 5, we see that T' must have a scatter at least $\ell' - \delta\sqrt{d}/2$.

It can be seen easily that the edges of T' are mapped to an Eulerian tour of the multigraph H constructed in step 7, and thus, step 8 succeeds. Again, we cannot guarantee that the recovered Eulerian tour is the same as the one to which T' maps. Any Eulerian tour of H, however, must respect the edge-multiplicities of H, which in turn are determined by the number of points that map to each vertex of H. Therefore, step 9 must succeed, and the output is YES.

Note. The restrictions on m and v in step 4 can be strengthened, resulting in a smaller number of iterations (and thus better running time). For instance, since each virtual edge corresponds to a hop via a point outside of B'_p, we could require the values of v to sum to $|P \setminus B'_p|$. We ignore such technicalities, as they do not affect the correctness of the algorithm – in the case of wrong values, we get the NO output in some of the later steps.

Running time. The cost of steps 1–3 is dominated by the cost of the loop starting in step 4. We observe that by Lemma 4(ii), $|C| \leq (9.8\sqrt{d}/\epsilon)^d$. Steps 5 and 6 amount to finding a perfect matching, and steps 7 and 8 amount to finding an Eulerian tour, both in a graph with $O(n)$ vertices. As for step 4, observe that the values of the functions m and v over all pairs in C sum to $|P \cap B'_p| \leq n$, so we need to consider at most $\binom{n}{|C|^2}$ ways of distributing a value of at most n into the integer values of m and v. Multiplying, and using a standard bound on the binomial, we obtain that the running time is at most $O(n^{|C|^2+3}) = O\left(n^{(100d/\epsilon^2)^d}\right)$.

This concludes the proof of Theorem 1.

References

[1] E. M. Arkin, Y. Chiang, J. S. B. Mitchell, S. Skiena, and T. Yang. On the maximum scatter TSP. In *SODA 1997*.

[2] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[3] A. I. Barvinok. Two algorithmic results for the traveling salesman problem. *Math. Oper. Res.*, 21(1):65–84, 1996.

[4] B. Bollobás. *Modern Graph Theory*. Springer, 1998.

[5] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Carnegie Mellon University, 1976.

[6] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problems project, 2012.

[7] S. P. Fekete. Simplicity and hardness of the maximum traveling salesman problem under geometric distances. In *SODA, 1999*.

[8] G. Gutin, A. Punnen, A. Barvinok, E. K. Gimadi, and A. I. Serdyukov. *The Traveling Salesman Problem and Its Variations*. Springer, 2002.

[9] M. Karpinski, M. Lampis, and R. Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015.

[10] L. Kowalik and M. Mucha. Deterministic -approximation for the metric maximum TSP. *Theoretical Comp. Sc.*, 410(4749):5000 – 5009, 2009.

[11] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM J. Comput*, 28:402–408, 1996.

[12] R. Parker and R. L. Rardin. Guaranteed performance heuristics for the bottleneck travelling salesman problem. *Oper. Res. Lett.*, 2(6):269–272, Mar. 1984.

[13] J.-R. Sack and J. Urrutia, eds. *Handbook of Computational Geometry*. North-Holland Publishing Co., 2000.

Approximating Multidimensional Subset Sum and the Minkowski Decomposition of Polygons *

Ioannis Z. Emiris † Anna Karasoulou † Charilaos Tzovas‡

Abstract

We consider the approximation of two NP-hard problems: Minkowski Decomposition (MinkDecomp) of integral lattice polygons, and the related Multidimensional Subset Sum (kD-SS). We prove, through a gap-preserving reduction, that, for general dimension k, kD-SS does not have an FPTAS. For $2D$-SS, we present an $O(n^7/\epsilon^4)$ approximation algorithm, where n is the set cardinality and ϵ bounds the approximation, and use it to approximate MinkDecomp.

1 Introduction

A polygon Q is called *lattice polygon* when all its vertices are integer points.

Problem 1 ***Minkowski Decomposition (MinkDecomp).*** *Given a lattice convex polygon Q, decide if it is decomposable, that is, if there are nontrivial lattice polygons A and B such that $A + B = Q$, where $+$ denotes Minkowski sum. The polygons A and B are called the* summands.

Problem 2 ***MinkDecomp-μ-approx***

Input: A lattice polygon Q and a parameter $0 < \epsilon < 1$ and μ is a measure of polygons.

Output: Lattice polygons A, B such that $\mu(Q) - \epsilon X < \mu(A+B) < \mu(Q) + \epsilon X$. We call such an output an ϵX-solution.

Problem 1 is proven NP-complete in [5] and can be reduced to $2D$-SS. For the reduction see Section 4.

Problem 3 ***kD-Subset Sum (kD-SS)***

Input: A vector set $S = \{v_i \mid v_i \in \mathbb{Z}^k, 1 \leq i \leq n\}$ and a target vector $t \in \mathbb{Z}^k$.

Output: Decide whether there exist a vector subset $S_\tau = \{v_1, v_2, \ldots, v_\tau\} \subseteq S$ such that $\sum_{i=1}^{\tau} v_i = t$.

*Ioannis Z. Emiris and Anna Karasoulou were partially supported by H2020 M. Sklodowska-Curie Innovative Training Network "ARCADES: Algebraic Representations in Computer-Aided Design for complEx Shapes", 2016-2019.

†Department of Informatics & Telecommunications, University of Athens, {emiris,akarasou}@di.uoa.gr

‡Department of Mathematics, University of Athens, tzovasx@math.uoa.gr

This is a generalization of the classic $1D$-SS problem, and as such, is also NP-complete.

Let P_i be the set of all possible vector sums that can be produced by adding at most i elements from the first i vectors in S. Then, P_n is the set of all possible vector sums. Here is the approximation version:

Problem 4 ***kD-SS-approx***

Input: A set $S = \{v_i \mid v_i \in \mathbb{Z}^k, 1 \leq i \leq n, k \geq 1\}$, a nonzero target $t \in P_n$ and $0 < \epsilon < 1$.

Output: Find a subset $S_\tau = \{v_1, v_2, \ldots, v_\tau\} \subseteq S$ whose vector sum t' satisfies $dist(t, t') \leq \epsilon|t|$, where $|t|$ is the length of t.

We consider Euclidean distance l_2 but the discussion is easily generalized to any l_p, $1 \leq p < \infty$.

Definition 1 *A PTAS (Polynomial Time Approximation Scheme) is an algorithm which takes an instance of an optimization problem and a parameter $\epsilon > 0$ and, in polynomial time, produces a solution that is within a factor $1 + \epsilon$ of being optimal for minimazation problems, or $1 - \epsilon$ for maximization. One further defines class FPTAS (Fully PTAS) where the time complexity is polynomial in both input size and parameter ϵ.*

Previous work $1D$-SS and kD-SS are not strongly NP-complete and can be solved exactly in pseudo-polynomial time: $1D$-SS is solved in $O(nt)$ and, generalizing this idea, kD-SS is solved in $O(n|M|^k)$, where $M = \max P_n$ is the farthest reachable point; information for $k = 2$ in [8]. Moreover, $1D$-SS is in FPTAS [6].

A similar problem to kD-SS is the Closest Vector Problem (CVP): we are given a set of basis vectors $B = \{b_1, \ldots, b_n\}$, where $b_i \in \mathbb{Z}^k$, and a target vector $t \in \mathbb{Z}^k$, and we ask what is the closest vector to t in the lattice $\mathcal{L}$ generated by B. The lattice of B is $\mathcal{L}(B) = \{\sum_1^m a_i b_i \mid a_i \in \mathbb{Z}\}$ and thus kD-SS is a special case of CVP, where $a_i \in \{0, 1\}$. CVP cannot be approximated within a factor of $2^{\log^{1-\epsilon} n}$ [1, 3].

MinkDecomp has its fair share of attention. One application is in the factorization of bivariate polynomials through their Newton polygons. If a polynomial factors, then its Newton polygon has a Minkowski decomposition. Here, we are interested in finding approximate solutions of the latter: polygons

whose Minkowski Sum is almost the original polygon. MinkDecomp is NP-complete for integral polygons, and a pseudopolynomial algorithm exists [5]. An algorithm for polynomial irreducibility testing using MinkDecomp is presented in [7]. They present a criterion for MinkDecomp that reduces the decision problem into a question in linear programming. Continuing the work in [4], we propose a poly-time algorithm that solves MinkDecomp approximately using a solver for $2D$-SS.

Our contribution We introduce the kD-SS problem. It is clearly NP-complete; we prove that it cannot be approximated efficiently. We design an algorithm for $2D$-SS-approx: given a set S, $|S| = n$, target t and $0 < \epsilon < 1$, the algorithm returns, in $O(n^7\epsilon^{-4})$, a subset of S whose vectors sum to t' such that $dist(t, t') \leq \epsilon M$, where M is the length of the largest possible sum of vectors in any subset of S. This algorithm yields an approximation algorithm for MinkDecomp: If Q is the input polygon the algorithm returns polygons A and B whose Minkowski sum defines polygon Q' such that $vol(Q) \leq vol(Q') \leq vol(Q) + \epsilon D^2$ and $per(Q) \leq per(Q') \leq per(Q) + 2\epsilon D$, where D is the diameter of Q, i.e., the maximum distance between two vertices of Q.

2 kD-SS is not in FPTAS

To prove that kD-SS is not in FPTAS we will apply the idea for the CVP, which is not in PTAS [1]. We will change their proof and apply it to our problem to prove something weaker for kD-SS.

Given a CNF formula ϕ we invoke Proposition 1 and get an instance of the Set Cover problem. This is a gap introducing reduction, because if ϕ is statisfiable then the instance of Set Cover has a solution of size exactly K and if ϕ is not statisfiable every solution has size at least cK for a constant c. From this instance of Set Cover we create an instance for kD-SS that preserves the gap. Now, if ϕ is satisfiable, the closest vector to a given target t has distance exactly K. If ϕ is not statisfiable, the closest vector in target t has distance at least cK.

We reduce kD-SS to Set Cover for norm l_1, but this can easily be generalized to any l_p, where p is a positive integer. We say that a cover is *exact* if the sets in the cover are pairwise disjoint.

Proposition 1 *[2] For every $c > 1$ there is a polynomial time reduction that, given an instance ϕ of SAT, produces an instance of Set Cover $\{\mathcal{U}, (S_1, \ldots, S_m)\}$ where $\mathcal{U}$ is the input set of integers and $S_1, \ldots, S_m$ are subsets of $\mathcal{U}$, and integer K with the property: If ϕ is satisfiable, there is an exact cover of size K, otherwise all set covers have size more than cK.*

Theorem 2 *Given a CNF formula ϕ and $c > 1$ we create an instance $\{v_1, \ldots, v_n, t\}$ of kD-SS. If ϕ is satisfiable, then the minimum distance from t is less than K or otherwise, it is more than cK.*

Proof. Let $\{\mathcal{U}, (S_1, \ldots, S_m), K\}$ be the instance of Set-Cover obtained in Proposition 1 for the formula ϕ. We transform it to an instance of kD-SS with input set S and target t, such that the distance of t from the nearest point in the set of all possible points P_n is either K or $\geq cK$.

Let v_i be the vectors of the reduction that will have $n+m$ coordinates, where $|\mathcal{U}| = n$. We will create such a vector v_i for every set S_i. Let $L = cK$. Then the first n coordinates of each vector v_i have their j'th-coordinate ($j \leq n$) equal to L, if the corresponding j'th-element belongs to set S_i, or 0 otherwise. The remaining m coordinates have 1 in the $(n + i)$'th-coordinate and zeros everywhere else:

$$v_i = (L \cdot \chi_{S_i}, 0, \ldots, 1, \ldots 0) = (L \cdot \chi_{S_i}, e_i)$$

The target vector t has in the first n coordinates L and the last m are zeros, $t = (L, \ldots, L, 0, \ldots, 0)$.

Now, let the instance of Set-Cover have an exact cover of size K. We will prove that the minimum distance from target t is less than K. Without loss of generality, name the solution $\{S_1, \ldots, S_K\}$. For each S_i, $1 \leq i \leq K$ sum the corresponding vectors v_i and let this be $\zeta \in \mathbb{Z}^{n+m}$:

$$\zeta = \sum_{i=1}^{K} v_i = (\underbrace{L, \ldots, L}_{n}, \underbrace{1, \ldots, 1}_{K} \underbrace{0, \ldots, 0}_{m-K}).$$

The first n variables must sum up to $(L, L \ldots, L)$, because if one of the coordinates was 0, the solution would not be a cover and if one of them was greater than L, then some element is covered more than once and the solution would not be exact. The key point is that in the last m coordinates we will have exactly K ones. The distance of this vector ζ from t is

$$\| - t + \zeta\|_1 = \|(\underbrace{0, \ldots, 0}_{n}, \underbrace{1, \ldots, 1}_{K} \underbrace{0, \ldots, 0}_{m-K})\|_1 = K$$

Thus, there is a point in P_n that its distance from t is at most K.

Let us consider the other direction, where the Set Cover instance has a solution set greater than $cK = L$. We will show that the closest vector in t has distance at least L. This solution must have at least $cK = L$ sets. As before, $\| - t + \zeta\|_1 \geq L$ (this time the cover need not be exact).

Towards a contradiction, suppose there exists a vector ξ such that $\| - t + \xi\|_1 < L$. If the corresponding sets do not form a cover of S then one of the first n coordinates of ξ is 0 and this alone is enough for $\| - t + \xi\|_1 > L$. If the sets form a cover that is not

exact, then in at least one of the first n coordinates of ξ will be greater than L (for the element that is covered more than once) and will force $\| - t + \xi\|_1$ to be greater than L. Finally, if the sets form an exact cover, the first n coordinates of $\| - t + \xi\|_1$ will be 0. For the distance to be less than L, in the last m coordinates there must be less than L units implying that the sets in the cover are less than L contradicting our hypothesis.

In all cases, there cannot exist a vector whose distance from t is $< cK$. □

Theorem 3 *There is no FPTAS for kD-SS-approx unless P=NP.*

Proof. ϕ is a given formula. Suppose there exists an FPTAS for kD-SS-approx. Run the algorithm with $\epsilon = 1/cn$ and target t. Since $\|t\|_1 = ncK$, we are looking for possible solutions within distance $\epsilon ncK = K$ from t. By Theorem 2, we distinguish whether ϕ is satisfiable or not in polynomial time. □

3 The approximation algorithm for $2D$-SS

In this section we discuss the approximation algorithm for $2D$-SS. The idea is to create all possible vectors step by step. At each step, if two vectors are close to each other, one is deleted. Whenever we refer to distance it is the Euclidean distance. We begin with notation.

- Input: the set $S = \{v_1, v_2, \ldots, v_n\}$ with $v_i = (x_i, y_i) \in \mathbb{Z}^2$ and $|S| = n$, parameter $0 < \epsilon < 1$.
- P_i is the set of all possible vectors that can be produced by adding at most i elements from the first i vectors in S. P_n is the set of all possible vector sums.
- $E_i = L_{i-1} \cup \{L_{i-1} + v_i\}$ is the list created at the beginning of every step and that is about to get trimmed.
- $L_i = trim(E_i, \delta)$ is the trimmed list.

At the beginning of the i-th step we create the list $E_i = L_{i-1} \cup \{L_{i-1} + v_i\}$. Notice that, addition is over $\mathbb{Z}^2$, and after a point is found we calculate its length and sort E_i based on the lengths. For each vector $u \in E_i$ with length $|u|$ and angle $\theta(u)$, check all the vectors $u' \in E_i$ that have length $|u| \le |u'| \le (1+\delta)|u|$. If also $\theta(u) - \delta \le \theta(u') \le \theta(u) + \delta$, remove u'. L_i is the trimmed list, that is, from the list E_i we remove vectors that are close to each other. The two conditions ensure that $dist(u', u) \le \alpha\delta|u|$, where $1 \le \alpha \le 2$ is a constant. Every vector that is deleted from E_i is not very far away from a vector in L_i:

$$\forall u \in E_i, \exists w \in L_i : u = w + r_w,\ |r_w| \le \alpha\delta|w| \quad (1)$$

hence, $|w| \le |u| \le (1+\delta)|w|$. See fig. 1.

Lemma 4 *Call function $L_i = \text{TRIM}(E_i, \delta)$ where E_i is an input list of vectors and $\delta = \epsilon/2n$ the parameter. Then $|L_i| = O(n^3\epsilon^{-2})$ for $1 \le i \le n$.*

Proof. Let $M_i = max\{|x_k| :\ x_k \in E_i\}$, the vector in E_i with the largest magnitude. Every vector in E_i has length between $(1+\delta)^k$ and $(1+\delta)^{k+1}$ that forms an annulus, called zone. Solving $(1+\delta)^k \ge M_i$ for k, there are $O(n^2/\epsilon)$ many zones.

Every zone can be divided in cells. Each cell is taken in such a way that it will cover $2\delta R$ of the lower circle, where R is the radius of the circle. Thus every zone has at most $2\pi R/\delta R = 4\pi n/\epsilon$ cells. List L_i has at most an entry for every cell created and size can be $|L_i| \le (n^2/\epsilon) \cdot (4\pi n/\epsilon) = O(n^3\epsilon^{-2})$. □

The running time for $2D$-SS-approx is $O(n|L_n|^2)$ and overall it requires time $O(n^7\epsilon^{-4})$.

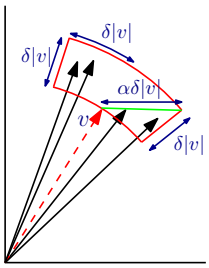


Figure 1: One cell for vector v.

Theorem 5 *For a set of vectors $S = \{v_i \mid v_i \in \mathbb{Z}^2, 0 \le i \le n\}$, every vector sum $v \in P_n$ can be approximated by a vector w*

$$\forall v \in P_n, \exists w \in L_n, \exists r_w : v = w + r_w,$$
$$|r_w| \le n\delta \max\{L_n\},$$

where $\max\{L_i\}$ is the length of the largest vector.

Proof. The proof is by induction. □

Setting $\delta = \epsilon/2n$ we can ensure that every possible vector sum will be approximated by a vector in L_n at most $\epsilon \max\{L_n\}$ far. Implementing and testing the algorithm, much better bounds are obtained.

Lemma 6 *Let $S = \{v_i \mid v_i \in \mathbb{Z}^2, 0 \le i \le n\}$ be the input set of vectors. If also all vectors are in one quadrant and for all $v_i \in S : |v_i| \ge \delta \sum_1^n |v_i|$ then*

$$\forall v \in P_n,\ \exists w \in L_n,\ \exists r_w : v = w + r_w,\ |r_w| \le n\delta|w|$$

Proof. The proof is by induction. □

In the special case, where all vectors are within an angle of 90 degrees and there are no short vectors, this algorithm gives an $(1+\epsilon)$-approximation solution meaning an FPTAS for $\delta = \epsilon/2n$.

4 Minkowski Decomposition using $2D$-SS

In this section we will describe an algorithm for approximating MinkDecomp. The algorithm takes an input polygon Q, transform it to an instance $\{S, t\}$ of $2D$-SS-approx and calls our algorithm for $2D$-SS-approx to solve MinkDecomp.

Let Q be the input to MinkDecomp: $Q = \{v_i \mid v_i \in \mathbb{Z}^2, 0 \leq i \leq n\}$. First we create the vector set U by subtracting successive vertices (in clockwise order): $U = \{v_1 - v_2, \ldots, v_n - v_0\}$. We call U the *edge sequence* and each vector is called an *edge vector*. Its edge sequence is denoted by $s(Q)$. For each edge vector v in $s(Q)$ we calculate its *primitive vector*.

Definition 2 *Let $v = (a, b)$ be a vector and $d = gcd(a, b)$. The* primitive vector of v is $e = (a/d, b/d)$.

We get an edge vector $(x, y) \in s(Q)$ and calculate its primitive vector $e = (x/d, y/d)$, where $d = gcd(x, y)$. The scalars $d_1, \ldots, d_k$ are computed by:

$$d_i = 2^i,\ i = 0, \ldots, \lfloor \log_2 d/2 \rfloor \text{ and } d_k = d - \sum_{i=1}^{\lfloor \log_2 d/2 \rfloor} d_i$$

We create the set S by adding the vectors $d_i e$ and repeat the procedure for all vectors $v \in s(Q)$. Notice that $\sum_1^k d_i = d$, so the primitive edge sequence also sums to $(0, 0)$. Using this construction, the primitive vectors added are $\log d$ for every $v \in s(Q)$. The primitive edge sequence uniquely identifies the polygon up to translation determined by v_0. This is a standard procedure as in [5, 4].

The main defect in this approach is that the algorithm returns a sequence of vectors $S' \subset S$ that sum close to $(0, 0)$ but possibly not $(0, 0)$. This means the corresponding edge sequence does not form a closed polygon. To overcome this, we just add to $s(A)$ the vector v, from the last point to the first, to close the gap. If $s(A)$ sums to a point (a, b), by adding vector $v = (-a, -b)$ to $s(A)$ the edge sequence $s(A) \cup \{v\}$ now sums to $(0, 0)$. If we rearrange the vectors by their angles, they form a closed, convex polygon that is summand A. We do the same for the sequence $s(B)$. Notice that the vector added in $s(B)$ is $-v = (a, b)$ and this sequence (rearranged) also forms a closed, convex polygon. We name $s(A') = s(A) \cup \{v\}$, $s(B') = s(B) \cup \{v\}$ and take their Minkowski Sum $Q' = A' + B'$, where A' and B' are the convex polygons formed by $s(A')$ and $s(B')$. We measure how close Q' is to the input Q. Let D be the diameter of Q, the maximum distance between two vertices of Q.

Lemma 7 *Let the summands A' and B' of Q', as discussed. We deduce that $vol(Q') \leq vol(Q) + \epsilon D^2$ and $per(Q') \leq per(Q) + 2\epsilon D$.*

Proof. We observe that

$$s(Q') = s(A') \cup s(B') = s(A) \cup s(B) \cup \{v, -v\} \implies$$
$$s(Q') = s(Q) \cup \{v, -v\}.$$

This equals adding to Q a single segment s of length $|s| = |v|$ and $Q' = Q + s$. Since $per(Q) = \sum_{v \in s(Q)} |v|$, it follows $per(Q') = per(Q) + 2|v|$. For the volume of Q', at worst, where v is perpendicular to D, a rectangle with sides v and D is added to Q. This extra volume is $\leq |v|D$, thus $vol(Q') \leq vol(Q) + |v|D$. It is also easily observed that $vol(Q) \leq vol(Q')$ and $per(Q) \leq per(Q')$ since $Q' = Q + s$.

The length of vector v we add to close the gap, is the key factor to bound polygon Q'. From the guarantee of the $2D$-SS-approx algorithm we know that $s(A)$ (and respectively $s(B)$) sum to a vector with length at most $\epsilon \max\{L_n\}$. This is vector v and thus $|v| \leq \epsilon \max\{L_n\}$. Since $\max\{L_n\} \leq D$, we get $|v| \leq \epsilon D$. This yields $per(Q) \leq per(Q') \leq per(Q) + 2\epsilon D$ and $vol(Q) \leq vol(Q') \leq vol(Q) + \epsilon D^2$. □

Corollary 8 *The proposed algorithm provides a $2\epsilon D$-solution for MinkDecomp-per-approx and an ϵD^2-solution for MinkDecomp-vol-approx.*

This algorithm is implemented in Python 3, it is openly available through Github and Sage[1] and tested for polygons with up to 100 vertices and $\epsilon \in [0.1, 0.5]$ giving much better errors than the bounds proven. For instances with 50 vertices and ϵ=0.2, the algorithm needs around 60 minutes to find the summands.

References

[1] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computer and System Sciences*, 54(2):317 – 331, 1997.

[2] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 294–304, New York, NY, USA, 1993. ACM.

[3] I. Dinur, G. Kindler, R. Raz, and S. Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003.

[4] I. Emiris and E. Tsigaridas. Minkowski decomposition of convex lattice polygons. In *Algebraic Geometry and Geometric Modeling*, Mathematics and Visualization, pages 217–236. Springer Berlin Heidelberg, 2006.

[5] S. Gao and A. Lauder. Decomposition of polytopes and polynomials. *Discrete and Computanional Geometry*, 26(1):89–104, 7 2001.

[6] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, Oct. 1975.

[7] D. Kesh and S. Mehta. Polynomial irreducibility testing through minkowski summand computation. In *Proceedings of the 20th Canadian Conf. on Comp. Geom.*, pages 43–46, 2008.

[8] G. Zirdelis. Manuscript on Minkowski decomposition of convex lattice polygons. Course Project. 2014.

[1] https://github.com/tzovas/Approximation-Subset-Sum-and-Minkowski-Decomposition

Fair and Square: Cake-Cutting in Two Dimensions

Erel Segal-Halevi Shmuel Nitzan Avinatan Hassidim Yonatan Aumann*

Abstract

A polygonal land-estate ("cake") has to be divided among n agents. The division should satisfy the following two requirements: (a) Each piece should have a pre-specified geometric *shape*, such as a square. (b) Each agent should receive a piece with a *value* above a given threshold. The value of a piece is defined as the integral of a given value-density function over the piece. Each agent has a possibly different value-density, yet each agent should agree that the value of his piece is above the fairness threshold. Each of the two requirements has been studied before on its own: the geometric requirement is common in polygon decomposition problems, and the value requirement is common in the classic economics problem known as "fair cake-cutting". Our research combines these requirements. We present algorithms for dividing a square cake in a way both *fair* (in value) and *square* (in shape). The value guarantee per agent is $\Theta(1/n)$, where the constants depend on the cake shape.

1 Introduction

Several people inherited a land-estate. How can they divide it fairly among them?

Geometric division. Division problems are abundant in computational geometry. A survey from 2000 [5] lists over 100 papers about different variants of such problems. A typical problem involves a given polygon C and a given family S of polygons (triangles, squares, rectangles, star-shapes, spirals, etc). C should be partitioned to several components which are elements of S (henceforth *S-elements*). The partition should satisfy such requirements as: minimizing the number of pieces, minimizing the total perimeter of the pieces, etc. Sometimes it is also required that the pieces have the same area, e.g. [2, 6]. But the value of land is much more than its shape and area. For example, a land-plot near the sea may have a very different value than a land-plot with exactly the same shape and area in the middle of the desert. Geometric partition problems do not handle such considerations.

Economic division. Division problems are also abundant in economics and social choice. The land division problem is often called *cake-cutting* [3, 8]. There, value considerations are of key importance. Moreover, economists acknowledge that different people have different valuations. One person may prefer the sea-shore while another person may prefer the mountains. Hence, value is defined on an agent-by-agent basis: each agent i has a bounded and integrable *value-density* function on the cake, $v_i : C \to \mathbb{R}$. The value of a piece X to agent i is defined as the integral of the value-density: $V_i(X) := \int_{x \in X} v_i(x)dx$. The V_i thus defined are *nonatomic measures*, so there are no atoms which cannot be fairly divided.

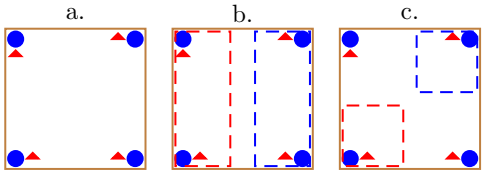


Figure 1: Dividing a square fairly to two agents.

An *allocation* of C is an n-tuple $X_1, \ldots, X_n$ of pairwise-disjoint subsets of C: $X_1 \cup \cdots \cup X_n \subseteq C$. An allocation is called *fair* or *proportional* if every agent is allotted a piece he values as at least $1/n$ the total cake value: $\forall i : V_i(X_i)/V_i(C) \geq 1/n$. Algorithms for finding fair allocations have been used since Biblical times. A famous algorithm for two agents is "cut and choose": the cake C is partitioned to two parts C', C'' which have the same value for Alice ($V_A(C') = V_A(C'') = V_A(C)/2$); the part more valuable to Bob is given to Bob and the other part is given to Alice. Thus both Alice and Bob are guaranteed a piece worth at least half their total cake value. This algorithm has been generalized to n agents in the 1940s [10] and many new algorithms have been published over the years [7]. But in contrast to the geometric partition problems, most of these algorithms do not pay much attention to the geometric shape of the pieces. Typically, C is assumed to be a 1-dimensional interval and the pieces are either intervals or a countable collection thereof (see the full paper [9] for some exceptions). While a 1-dimensional division can be projected on a two-dimensional cake, the resulting pieces might be long and narrow slivers that are unusable in practice.

Our division. We claim that both geometric shape and fair value are important. The input in our problem is a polygonal cake C, a family S of polygons

*Bar-Ilan University, Ramat-Gan, Israel. {erelsgl,shmuelnitzan,avinatanh,yaumann}@gmail.com.
Research is partially funded by ISF grant 1083/13, Doctoral Fellowships of Excellence Program and Mordecai and Monique Katz Graduate Fellowship Program at Bar-Ilan University.

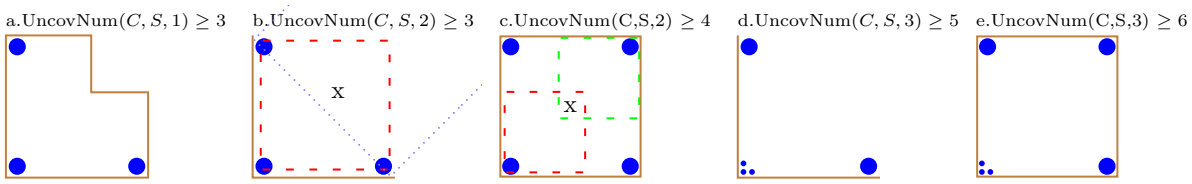


Figure 2: Uncover numbers of various cakes. S is the family of squares.

and n nonatomic value-measures $V_1, \ldots, V_n$. We are looking for *S-allocations* - allocations of C in which $\forall i : X_i \in S$. A simple example, illustrated in Figure 1, shows that the ideal of fairness, giving each agent at least $1/n$ of the total cake value, cannot always be satisfied. Here C is square and there are $n = 2$ agents, Disc and Triangle. The value-density of the Disc agent is 1 inside the discs and 0 outside; the value-density of the Triangle agent is 1 inside the triangles and 0 outside (a). When S is the family of rectangles, it is easy to give each agent $1/2$ of the total value (b). When S is the family of squares, it is easy to give each agent $1/4$ of the total value (c), but impossible to give both agents more than $1/4$. This motivates the following:

Definition 1 ***Prop***(C, S, n) *is the largest proportion* $f \in [0, 1]$ *such that, for every set of* n *nonatomic value-measures* $(V_1, \ldots, V_n)$*, there exists an* S*-allocation* $(X_1, ..., X_n)$ *of* C *for which* $\forall i : V_i(X_i)/V_i(C) \geq f$.

One-dimensional cake-cutting algorithms, e.g. [4], imply that $\text{Prop}(Interval, Intervals, n) = 1/n$. By a projection argument, this also implies that $\text{Prop}(Rectangle, Rectangles, n) = 1/n$. As we have just seen (Figure 1), $\text{Prop}(Square, Squares, 2) \leq 1/4$.

$\text{Prop}(C, S, n)$ is a purely geometric function: its value depends only on n and the geometric shapes of C and S. Intuitively, it describes how well the family S can be used to fairly divide C. This function is the focus of our research. We study many different combinations of C and S. For every such combination, we look for impossibility results like Figure 1 proving upper bounds on Prop, and division algorithms proving lower bounds on Prop. In the present abstract we illustrate some of our methods focusing on several simple cases: the cake C is a square, a quarter-plane or an unbounded plane (where the value-density functions always have a bounded support), and S is the family of squares. Section 2 presents impossibility results and Section 3 presents division algorithms. Many other combinations are described in [9], including arbitrarily-shaped cakes and arbitrary fat pieces.

2 Impossibility results

The key geometric tools used in our impossibility results are *uncovers* and *uncover-numbers*. They generalize the *anti-squares* studied e.g. by [1], who also show the duality between them and square-covers.

Definition 2 *(a) Let I be a set of discs contained in C. I is called an* ***n-S-uncover*** *in C if in any set of n pairwise-disjoint S-elements contained in C, at least one S-element overlaps only at most one disc of I.*

(b) The ***n-S-uncover number*** *of C, $\text{UncovNum}(C, S, n)$, is the maximum cardinality of an n-S-uncover in C.*

Some examples are illustrated in Figure 2. In (a), C is a rectilinear hexagon. The three discs are a 1-square-uncover, because any 1 square contained in C overlaps at most one disc. In (b), C is a quarter-plane and the three discs are a 2-square-uncover: any square that overlaps two or more discs must contain the "x" in its interior. Hence, in any set of 2 disjoint squares contained in C, at least one square overlaps at most one disc. Similarly in (c), C is a square and the four discs are a 2-square-uncover.

New uncovers can be constructed from existing ones using *deflation*. Let I_K and I_M be two copies of the 2-square-uncover of (b). Remove the bottom-left disc of I_M and deflate I_K such that all its three discs are contained in the previous location of that bottom-left disc. The result is the set of 5 discs in (d). We claim that it is a 3-square-uncover: there is at most one square overlapping two discs of the deflated I_K and at most one square overlapping two discs of I_M, so all in all there are at most two disjoint squares overlapping two discs of the arrangement in (d).

In general, we can prove the following *Deflation Lemma*: if I_K is a k-S-uncover containing K discs and I_M is an m-S-uncover containing M discs, then (under certain conditions that we omit here) it is possible to deflate I_K into I_M to get a $(k + m - 1)$-S-uncover containing $K + M - 1$ discs. This lemma allows us to construct (d) from (b) (with $m = k = 2$ and $M = K = 3$) and to construct (e) from (b)+(c) ($m = k = 2$ and $M = 4$ and $K = 3$).

Let I_M be an m-S-uncover with M discs. By recursively applying the Deflation Lemma, we can (under certain conditions) deflate I_M into one of its own discs to get uncovers with as many discs as we want. For every $n \geq 1$, it is possible to get an $(n-1)(m-1)+1$-S-uncover having $(n - 1)(M - 1) + 1$ discs. Tak-

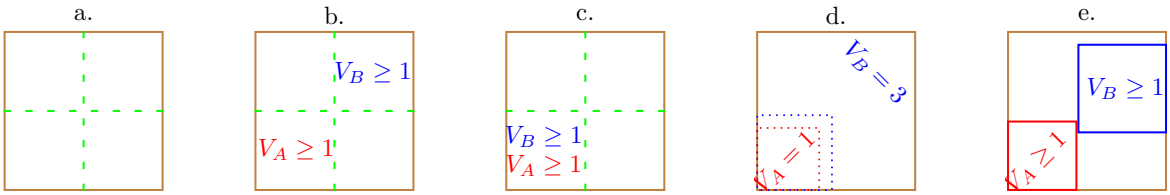


Figure 3: Division algorithm for dividing a square cake to 2 agents who want square pieces.

ing I_M to be the 2-square-uncover in (b), we build, for every $n \geq 2$, an n-square-uncover I_{2n-1} having $2n-1$ discs in a quarter-plane, proving that $\forall n \geq 1$: $\text{UncovNum}(QuarterPlane, Squares, n) \geq 2n-1$.

Deflating the set I_{2n-3} into the bottom-left disc of (c) gives, for every $n \geq 2$, an n-square-uncover having $2n$ discs in a square. This proves that $\forall n \geq 2$: $\text{UncovNum}(Square, Squares, n) \geq 2n$.

The following lemma links the uncover numbers to fair cake-cutting:

Lemma 1 *For every cake C, family S and $n \geq 1$:*

$$Prop(C, S, n) \leq 1/UncovNum(C, S, n)$$

Proof. Let $m = \text{UncovNum}(C, n, S)$ and let I_m be an n-S-uncover of cardinality m in C. Assume that the cake C is a desert and the elements of I_m are water-pools. Assume that all n agents have the same value-density function, which assigns a value of 1 to each pool and is 0 outside the pools. By Definition 2, in every allocation of n S-elements, at least one S-element overlaps at most one pool. The agent receiving this piece has a value of at most $1 = V(C)/m$. □

With the uncover numbers from above, we get:

Corollary 2 *a. $Prop(Square, Squares, n) \leq 1/(2n)$; b. $Prop(QuarterPlane, Squares, n) \leq 1/(2n-1)$.*

3 Division algorithms

The key geometric tools used in our positive results are *covers* and *cover numbers*.

Definition 3 *(a) An **S-cover of** C is a set of S-elements whose union equals C.*
*(b) The **S-cover number** of C, $CoverNum(C, S)$, is the minimum cardinality of an S-cover of C.*

For example, it is easy to see that for the hexagon in Figure 2/a, $\text{CoverNum}(C, Squares) = 3$, since it can be covered by a set of 3 squares (and no set of 1 or 2 squares can cover it).

Lemma 3 *For every cake C and family S:*

$$Prop(C, S, 1) \geq 1/CoverNum(C, S)$$

Proof. By the pigeonhole principle, if there is an S-cover of C with m elements, then one of these elements must have a value of at least $V(C)/m$. □

We now present an introductory division algorithm. C is a square, S the family of squares and there are $n = 2$ agents, Alice and Bob. Without loss of generality, we scale the value-densities of the agents such that the value of the entire cake is exactly 4 for both of them. By the example of Figure 1 and by Corollary 2, we already know that the largest value that can be guaranteed to both agents is 1. Our algorithm indeed guarantees each agent a value of at least 1.

The algorithm is illustrated in Figure 3 and it proceeds as follows. (a) Partition C to 4 quarters in a 2×2 grid and calculate the value of each quarter according to each agent. For each agent, select a quarter with a value of at least 1 (such a quarter must exist by the pigeonhole principle). (b) If the selected quarters are different, then give each agent his/her selected quarter and finish. Here it is easy to satisfy both agents since each agent prefers a different geographic region - Alice likes the south-west and Bob likes the north-east. (c) If the selected quarter is the same, then (d) for each agent, mark inside the selected quarter, a corner-square with a value of exactly 1 for that agent. (e) Cut a corner-square between the two marks. Give it to the agent associated with the smaller mark (Alice, in this case); that agent obviously receives a value of at least 1. For the other agent (Bob), the value of the remaining L-shape is at least 3. Cut a square from the remaining L-shape with a value of at least 1 (such a square must exist by Lemma 3) and give it to Bob. Note that Bob's square is larger than Alice's square; this makes sense, since Alice won a square in the south-west, which, according to both agents, is a valuable region. So Bob is compensated by winning a larger plot.

This algorithm proves $\text{Prop}(Square, Squares, 2) \geq 1/4$, which matches the upper bound of Corollary 2.

There are several ways to generalize this algorithm to n agents. We present here high-level sketches of the algorithms and refer the reader to [9] for more details.

Algorithm 1. For dividing a square cake, we use a "recursive quartering" technique. The cake is partitioned to four quarters as in Figure 3/a. The value of each quarter according to each agent is calculated.

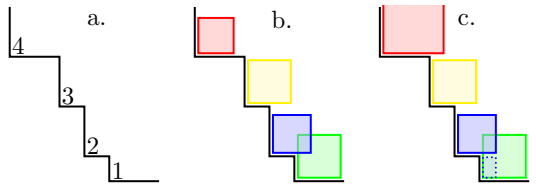


Figure 4: Dividing a 4-staircase.

The agents are assigned to quarters based on their values: each agent is assigned to a quarter which is more valuable to that agent (considering the number of other agents assigned to that quarter). Then, each quarter is divided recursively to the agents assigned to it (handling some special cases which we omit here). If done correctly, this algorithm guarantees to each agent a value of at least 1, when the value of the original cake is normalized to $4n-4$. This proves that: $\text{Prop}(Square, Squares, n) \geq 1/(4n-4)$. This has a multiplicative gap of 2 from the upper bound of Corollary 2. We know how to close this gap in the special case of n agents with identical value measures.

Algorithm 2. For dividing a quarter-plane cake, we use a "staircase carving" technique. We want to match the upper bound of Corollary 2 which is $1/(2n-1)$. To do this, we generalize and handle a cake in the shape of a *staircase*. A staircase is a polygonal domain which is bounded in two sides (e.g. left and bottom) but open in the other two sides. A *k-staircase* is a staircase that has k inner corners (see Figure 4/a). A quarter-plane is a 1-staircase. In every k-staircase, it is easy to find an n-square-uncover with $2n-2+k$ discs, so by Lemma 1, $\text{Prop}(k\,staircase, Squares, n) \leq 1/(2n-2+k)$. The following algorithm matches this bound.

Assume that n agents (with different value-densities) value the k-staircase as $2n-2+k$. For each corner j and agent i, mark a corner-square in corner j with a value of exactly 1 for agent i. In each corner, keep only one smallest square. There are two cases. *Easy case*: at least one square is entirely contained in its corner (like the square in corner 4 in Figure 4/b). Cut this square and give it to its agent. The remaining cake is a $(k+1)$-staircase and its value for the remaining $n-1$ agents is at least $2n-3+k = 2(n-1)-2+(k+1)$ so we can divide it to them recursively. *Hard case*: all squares flow over their corners, casting "shadows" on the corners above and/or to their right. We cannot just cut a square because the result will not be a staircase. Fortunately, we can prove the following geometric lemma: there always exists a square whose shadows are entirely contained in the other squares (like the square in corner 2 in Figure 4/c). Cut this square, give it to its agent, and discard its shadow/s. The value of each shadow to the other agents is at most 1; each removed shadow removes one corner and decreases k by 1; hence, the remaining $n-1$ agents still value the remaining cake as at least $2(n-1)-2+k'$ (where k' is the new number of corners) and we can proceed recursively.

This shows that $\text{Prop}(k\,staircase, Squares, n) = 1/(2n-2+k)$. By letting $k=1$ we get a tight result: $\text{Prop}(QuarterPlane, Squares, n) = 1/(2n-1)$.

Algorithm 3. Assume that C is an unbounded plane. Similarly to Algorithm 1, partition it to four quarter-planes and partition the agents to four groups according to their valuations. Then, divide each quarter-plane to its agents using Algorithm 2. A calculation in [9] gives, for all $n \geq 4$: $\text{Prop}(Plane, Squares, n) \geq 1/(2n-4)$. The table below summarizes the results presented in this abstract:

Cake	Lower	Upper
Square	$1/(4n-4)$	$1/(2n)$
1/4-plane	$1/(2n-1)$	$1/(2n-1)$
Plane	$1/n$	$1/(2n-4)$

For an unbounded plane we do not have an upper bound (other than the trivial upper bound of $1/n$). Hence, we conclude with an open question: Is it possible to divide an unbounded plane such that each of n agents receives a square piece worth at least $1/n$? Is it possible to divide the plane "fair and square"?

References

[1] M. O. Albertson and C. J. O'Keefe. Covering Regions with Squares. *SIAM Journal on Algebraic Discrete Methods*, 2(3):240–243, Sept. 1981.

[2] P. V. M. Blagojević and G. M. Ziegler. Convex equipartitions via Equivariant Obstruction Theory. *Israel Journal of Mathematics*, 200(1):49–77, 2014.

[3] S. J. Brams and A. D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, Feb. 1996.

[4] S. Even and A. Paz. A note on cake cutting. *Discrete Applied Mathematics*, 7(3):285–296, Mar. 1984.

[5] J. M. Keil. Polygon Decomposition. In *Handbook of Computational Geometry*, pages 491–518. University of Saskatchewan Saskatoon, Sask., Canada, 2000.

[6] R. Nandakumar and R. Rao. 'Fair' Partitions of Polygons - an Introduction. *Proceedings - Mathematical Sciences*, 122(3):459–467, Nov. 2010.

[7] A. D. Procaccia. Cake Cutting Algorithms. In Brandt, Conitzer, Endriss, Lang and Procaccia, editors, *Handbook of Computational Social Choice*, chapter 13. Cambridge University Press, 2015.

[8] J. M. Robertson and W. A. Webb. *Cake-Cutting Algorithms: Be Fair if You Can*. A K Peters/CRC Press, first edition, July 1998.

[9] E. Segal-Halevi, S. Nitzan, A. Hassidim, and Y. Aumann. Fair and Square: Cake-Cutting in Two Dimensions, Oct. 2015. arXiv preprint 1510.03170.

[10] H. Steinhaus. The problem of fair division. *Econometrica*, 16(1):101–104, Jan. 1948.

Voronoi Diagrams for Parallel Halflines in 3D

Franz Aurenhammer* Günter Paulini† Bert Jüttler‡

Abstract

We consider the Euclidean Voronoi diagram for a set of n parallel halflines in $\mathbb{R}^3$. A relation of this diagram to planar power diagrams is shown, and is used to analyze its geometric and topological properties. Moreover, a simple plane-sweep algorithm is given that computes the Voronoi diagram for parallel halflines at logarithmic cost per face.

1 Introduction

The Voronoi diagram is a powerful and widely used geometric partitioning structure. Many of its properties are well understood, also in generalized settings of various kinds; see e.g. [5].

Still, knowledge becomes quite sparse in dimensions larger than two, when sites of more general shape are allowed. This concerns the structural as well as the algorithmic properties, and is already true for the generalization from point sites to line segments. The combinatorial complexity of the Voronoi diagram for n line segments, and in particular, for n straight lines in Euclidean d-space $\mathbb{R}^d$ can be as large as $\Omega(n^{d-1})$; see [3]. The only known upper bound follows from a general result on lower envelopes of hypersurfaces [12], and is $O(n^{d+\varepsilon})$ for any $\varepsilon > 0$.

Even in $\mathbb{R}^3$, no better bounds than $\Omega(n^2)$ and $O(n^{3+\varepsilon})$, respectively, are known up to date. This may be partially due to the complicated shape of the arising bisector surfaces. They contain, among other components, parabolic and hyperbolic patches, and can lead to a diagram of fairly complicated topological structure. Already for three straight lines as sites, the induced structure gets so intricate that a separate paper has been devoted to its exploration [8].

To make the problem more tractable, several restricted scenarios have been considered. For example, if the line segment sites are confined to have constantly many orientations [10], then the size of the diagram reduces to $O(n^{2+\varepsilon})$. If, on the other hand, the underlying distance function is polyhedral and convex, then the diagram becomes piecewise-linear. The upper bound then can be tightened to $O(n^2\alpha(n)\log n)$, even when constant-sized convex polyhedra are allowed as sites; see [6] and [9], respectively. A practical algorithm for computing the medial axis of a nonconvex polytope in $\mathbb{R}^3$ under a convex polyhedral distance function is given in [2].

In the present note, we discuss a simple though nontrivial special case for the Euclidean distance, namely, the case where all sites are parallel halflines in $\mathbb{R}^3$, being unbounded in the same direction. Apart from the theoretical interest, practical applications arise in certain problems in the drilling industry (mining exploitation, offshore drilling, hydraulics, etc.), as is reported by Adamou [1]. In particular, such Voronoi diagrams serve in the exploration of the nearest layers to avoid collision between wells and identifying unwanted plies. A related problem where this diagram may be useful is approximate nearest-neighbor searching among a set of parallel line segments in $\mathbb{R}^3$, which has been studied in Emiris et al. [7].

As an interesting fact, the Voronoi diagram for parallel halflines is related to planar power diagrams. We describe this correspondence in Section 2, along with its structural implications. On the algorithmic side, a simple plane-sweep algorithm is obtained in Section 3. Basically, a power diagram for fixed sites has to be updated under continuous changes of site weights. Section 4 studies the behavior of the trisector curves for the halfline Voronoi diagram, motivated by an attempt to reduce the $O(n^{2+\varepsilon})$ upper bound on its combinatorial complexity (which follows from the result in [10]) to $O(n^2)$. Some extensions of our results are mentioned in Section 5.

2 Diagram

Let $H = \{h_1, \ldots, h_n\}$ be a set of parallel halflines in $\mathbb{R}^3$. We assume that each h_i is vertical, and unbounded in negative z-direction. The upper endpoint of h_i is denoted by z_i. We call z_i the *tip* of h_i, and (by slight abuse of notation) we will use z_i also to denote the z-coordinate of the tip. The distance of a point $x \in \mathbb{R}^3$ to a halfline h_i is defined as

$$d(x, h_i) = \min\{\delta(x, q) \mid q \in h_i\}$$

where δ denotes the Euclidean distance function. This distance is the normal distance of x to the supporting

*Institute for Theoretical Computer Science, University of Technology, Graz, Austria, auren@igi.tugraz.at

†Institute for Theoretical Computer Science, University of Technology, Graz, Austria, guenter.paulini@igi.tugraz.at

‡Institute of Applied Geometry, Johannes Kepler University, Linz, Austria, bert.juettler@jku.at

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

line, ℓ_i, of h_i, unless $d(x, h_i)$ is attained at the tip z_i. The region of a halfline in the Voronoi diagram, $\mathcal{V}(H)$, of H is given by

$$\mathrm{reg}(h_i) = \{x \in \mathbb{R}^3 \mid d(x, h_i) \leq d(x, h_j), \text{for all } j\}.$$

Regions are bounded by bisectors, B_{ij}, for pairs of halflines h_i, h_j. Let the respective tips satisfy $z_i \geq z_j$. Then B_{ij} is composed of three parts: A planar patch, contained in the (vertical) bisecting plane of the lines ℓ_i and ℓ_j, a piece of a parabolic cylinder equidistant from line ℓ_i and point z_j, and another planar patch in the bisecting plane of the points z_i and z_j. In the case $z_i = z_j$, B_{ij} is a single vertical plane.

The analysis of the structure of $\mathcal{V}(H)$ is eased by the fact that the generators of the parabolic patches are horizontal lines. This gives the following property:

Observation 1 *The intersection of B_{ij} with any horizontal plane is a straight line.*

Denote with E^Δ the horizontal plane $z = \Delta$, and consider the lines $b_{ij} = B_{ij} \cap E^\Delta$. As bisectors intersect 3 by 3 in trisectors $t_{ijk} = B_{ij} \cap B_{ik} \cap B_{jk}$, the lines b_{ij}, b_{ik}, and b_{jk} concur in a common point (or are parallel), for any pairwise different indices i, j, k. This implies, by a result in [4], that the line system $(b_{ij})_{1 \leq i < j \leq n}$ is the set of *power lines* defined by n weighted points in E^Δ. A more direct argument follows from the lemma below.

Lemma 1 *Consider the point $p_i = \ell_i \cap E^\Delta$, and assign the weight $w_i = -\max\{0, (\Delta - z_i)\}^2$ to it. For any $x \in E^\Delta$, we have $d(x, h_i)^2 = \delta(x, p_i)^2 - w_i$.*

Proof. If E^Δ lie below z_i then $p_i \in h_i$ and $w_i = 0$, and the assertion is trivial. Otherwise, it follows from the Pythagorean theorem, because h_i is normal to E^Δ. □

In other words, the squared distance of x to h_i is the power distance of x to the point p_i with weight w_i. We therefore have the following geometric relation:

Theorem 2 *For all values Δ, the sectional diagram $\mathcal{V}(H) \cap E^\Delta$ is identical to the power diagram of the points $p_1, \ldots, p_n$, for the weights w_i in Lemma 1.*

In particular, if E^Δ lies below all tips then the Euclidean Voronoi diagram of $p_1, \ldots, p_n$ is obtained.

Figure 1 displays the trisector arcs of $\mathcal{V}(H)$ for a set H of 10 halflines. A corresponding sectional power diagram is shown in Figure 2.

Theorem 2 indicates that $\mathcal{V}(H)$ must have a relatively simple structure, which we will study now in more detail. First of all, the weights w_i, when seen as functions $w_i(\Delta)$, are continuous. $w_i(\Delta)$ is zero for $\Delta \leq z_i$, and decreases quadratically for $\Delta > z_i$.

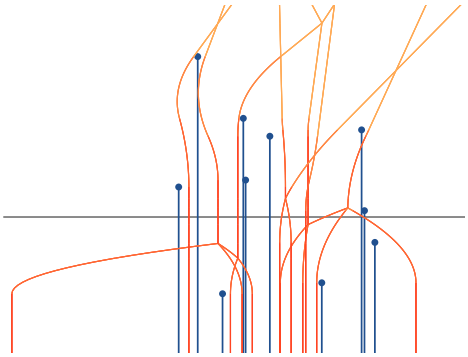


Figure 1: A halfline diagram with sectional plane, projected normal to the z-axis.

We watch the interplay on E^Δ when Δ is increased from $-\infty$ to ∞. The power cells $C_i(\Delta) = \mathrm{reg}(h_i) \cap E^\Delta$ are convex polygons, whose vertices move continuously. For sufficiently small Δ, each cell $C_i(\Delta)$ is a planar Voronoi region, and therefore is non-empty. Its edges first poise, and then move self-parallely because $p_1, \ldots, p_n$ stay fixed, and movement is in a fixed direction by the shape of the bisectors B_{ij}. So each point $x \in E^\Delta$ can enter or leave $C_i(\Delta)$ at most once. Also, if $C_i(\Delta)$ disappears from the diagram it cannot reappear, by the monotone movement of its edges. We summarize:

Property 1 *The intersection of $\mathrm{reg}(h_i)$ with every vertical line is connected or empty. Moreover, $\mathrm{reg}(h_i)$ is a simply-connected set.*

Note that a power cell $C_i(\Delta)$ survives for $\Delta \to \infty$ if and only if the tip z_i appears on the upper convex hull of $\{h_1, \ldots, h_n\}$. Property 1 does not imply that the combinatorial size of $\mathrm{reg}(h_i)$ is $O(n)$: Although the number of bisectors B_{ij} that border $\mathrm{reg}(h_i)$ is trivially limited to $n - 1$, a single bisector may define more than one *facet* (connected boundary patch) of $\mathrm{reg}(h_i)$. Indeed, there are multiple adjacencies between the regions in $\mathcal{V}(H)$ in general; see Section 4.

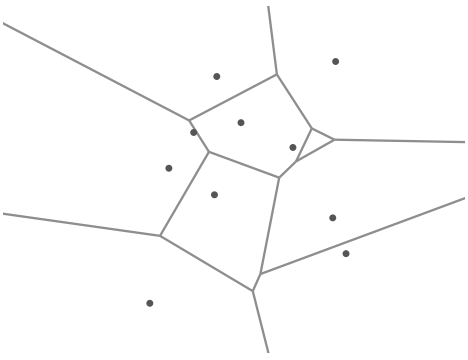


Figure 2: The sectional power diagram.

Let us now have a look at the Voronoi diagram $\mathcal{V}(\{h_i, h_j, h_k\})$ for only three halflines. The trisector curve t_{ijk} corresponds to a power diagram vertex $u^\Delta = t_{ijk} \cap E^\Delta$ for all Δ, unless the points p_i, p_j, and p_k are collinear (which we will exclude for the ease of description). This implies:

Property 2 *Each trisector t_{ijk} is a connected curve, unbounded in both z-directions, and monotone.*

In particular, t_{ijk} does not contain cycles. For pairwise different tip heights, the curve t_{ijk} is composed of 4 pieces, as can be easily verified: a halfline, two quadratic arcs, and another halfline. Therefore the algebraic degree of t_{ijk} is only two. Still, trisectors show a complicated intersection pattern in general. We will address this issue in Section 4.

3 Algorithm

Theorem 2 suggests a plane-sweep algorithm that computes the diagram $\mathcal{V}(H)$ in ascending z-direction.

The task is to maintain a power diagram for fixed points in the plane, under variation of their weights. The incidence structure of $\mathcal{V}(H)$ then can be inferred from the combinatorial changes that take place in the power diagram: When a power diagram edge appears (or disappears, respectively), then a facet of $\mathcal{V}(H)$ is born (or completed). Moreover, the collapse of a power cell signals the completion of a region in $\mathcal{V}(H)$.

An entirely two-dimensional implementation has been done, which avoids computing (costly) intersections of three-dimensional bisectors. Once the combinatorial structure of $\mathcal{V}(H)$ has been extracted, the bisector patches and trisector arcs that determine the geometry of $\mathcal{V}(H)$ are calculated in a final step.

To describe the combinatorial part of the algorithm in more detail, let $\mathrm{PD}(\Delta)$ be the power diagram for the points $p_1, \ldots, p_n$ with weights $w_1(\Delta), \ldots, w_n(\Delta)$, as defined in Section 2. We start with any value $\Delta < \min\{z_1, \ldots, z_n\}$, and initialize $\mathrm{PD}(\Delta)$ as the planar Voronoi diagram of $\{p_1, \ldots, p_n\}$.

There is only one type of *events* (z-values) where the power diagram can change. These are the anticipated life ends a_{ij} of its edges e_{ij}.

More specifically, a_{ij} is the z-value of the lowest intersection point above E^Δ of the respective two trisector curves t_{ijk} and t_{ijm}, which define the endpoints of e_{ij}. This value can be calculated in $O(1)$ time, by solving a quadratic equation in z for each of the intervals given by z_i, z_j, z_k, z_m. In the diagram $\mathrm{PD}(a_{ij})$, an update of constant complexity has to be performed. This update is either a flip that replaces the edge e_{ij} by the edge e_{km} (and a facet of $\mathcal{V}(H)$ in B_{ij} gets completed), or a collapse of a triangular cell incident to the edge e_{ij}, say $C_i(a_{ij})$ (and the region $\mathrm{reg}(h_i)$ gets completed).

The tips z_i of the halflines h_i do not lead to combinatorial changes in $\mathrm{PD}(z_i)$. They only alter the speeds of the edges in the power cell $C_i(z_i)$. This information is already incorporated in the trisector intersection task above.

We use a priority queue organized by z-values to maintain the order of events. Only $O(n)$ entries need to be stored at a time, by the linear number of edges in the power diagram $\mathrm{PD}(\Delta)$. The next event to be performed then is accessible in $O(\log n)$ time. Moreover, the total number of entries a_{ij} is bounded by the number of facets of $\mathcal{V}(H)$.

Note finally that the numbers of facets, arcs, and nodes of $\mathcal{V}(H)$ are linearly related: A region with f facets has $O(f)$ arcs and nodes, because the degree of its nodes is at least 3. We conclude:

Theorem 3 *$\mathcal{V}(H)$ can be computed in logarithmic time per face, using $O(n)$ extra storage.*

4 Trisectors

The combinatorial size of $\mathcal{V}(H)$ tends to be near-linear for many data, as has been observed in our experiments. Thus the output-sensitive algorithm in Section 3 can be expected to run fast in practice. On the other hand, $\mathcal{V}(H)$ can attain a complexity of $\Omega(n^2)$, for example, when the tips z_i are arranged like in a worst-case example for the Voronoi diagram of point sites in $\mathbb{R}^3$. This almost matches the upper bound of $O(n^{2+\varepsilon})$ for $\mathcal{V}(H)$, which follows from the more general bound in [10]; see Section 1. Proving a possible quadratic upper bound is complicated by the fact that the trisector curves of $\mathcal{V}(H)$ do not behave like pseudo-lines. Let us briefly comment on this fact.

For the halfline h_i with lowest tip, its region is always convex; all the bisectors B_{ij} either 'bend' towards h_i or are vertical planes. If the size of $\mathrm{reg}(h_i)$ can be shown to be $O(n)$, then an insertion argument for regions in ascending order of tip heights implies an overall $O(n^2)$ diagram size. Unfortunately, the result in [11] on the linear size of surface envelopes does not apply, because two trisector curves can intersect in more than one point.

To see an example, consider four halflines h_1, h_2, h_3, and h_4 arranged as is illustrated in Figure 3, from the top view (left) and from the front view (right). The two trisector curves t_{123} and t_{234} (and two others) concur in a point x, if and only if there exists a sphere centered at x that simultaneously touches all four halflines. There are two such spheres, a smaller one resting on the tip of the rightmost halfline, and bigger one passing through all four tips.[1]

[1] Thanks go to Peter Widmayer's group for pointing us to this example.

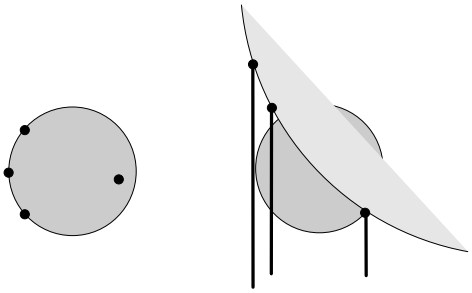


Figure 3: Two touching spheres for 4 halflines.

The trisectors defined by 4 halflines can have at most 3 intersection points, by a simple algebraic case analysis. This bound is actually attained, and even worse, there are constellations of n halflines for any $n \geq 4$ where every quadruple of related trisectors shows such an intersection behavior.

As another approach, one can try to bound the overall number of edges that appear in the power diagram $\mathrm{PD}(\Delta)$ for varying Δ. There are $\binom{n}{2}$ potential power edges. However, once having disappeared, an edge between the same two power cells can appear again. In fact this can happen $n-2$ times, which is the maximum possible. Stated differently, a fixed bisector B_{ij} can define $\Theta(n)$ facets where the two regions $\mathrm{reg}(h_i)$ and $\mathrm{reg}(h_j)$ are adjacent.

On the other hand, edge speeds in $\mathrm{PD}(\Delta)$ are not arbitrary. Starting with 0, the speed of an edge increases at constant acceleration, until it stays constant forever.

By the relationship between power diagrams and convex hulls (see e.g. [5]), the problem above can be transformed into a dynamic convex hull problem in $\mathbb{R}^3$. Starting from the paraboloid of revolution $z = x^2 + y^2$ at different times, n points move upwards vertically and at constant accelerations. The question of interest is now to bound the number of combinatorial changes that occur on their convex hull.

5 Extensions

An obvious extension of the results in this note concerns the Voronoi diagram of parallel line segments that are bounded in *both* directions. Whereas Lemma 1 can be generalized straightforwardly such that Theorem 2 still holds, the resulting plane-sweep algorithm now has to deal with the detection of new regions, which cannot be done locally. A simple solution is to calculate the respective Δ-values directly and beforehand, in $O(n \log n)$ time each.

Theorem 4 *The Voronoi diagram of n parallel line segments in $\mathbb{R}^3$ can be computed in $O((n^2+K)\log n)$ time and optimal space, where K denotes the size of the output.*

Our results also generalize to higher dimensions. For example, for computing the Voronoi diagram of parallel line segments in $\mathbb{R}^4$, a power diagram in $\mathbb{R}^3$ can be maintained. The sweep algorithm then constructs the desired diagram 2-face by 2-face and retains its output-sensitivity, though details get more involved.

References

[1] I. Adamou. Curvas y superficies bisectrices y diagrama de Voronoi de una familia finita de semirrectas paralelas end R^3. Ph.D. thesis, Department of Mathematics, University of Cantabria, Spain, 2013.

[2] O. Aichholzer, W. Aigner, F. Aurenhammer, and B. Jüttler. Exact medial axis computation for triangulated solids with respect to piecewise linear metrics. *Proc. Curves and Surfaces 2011*, J.-D. Boissonnat et al. (eds.), Springer Lecture Notes in Computer Science 6920, 2011, 1–27.

[3] B. Aronov. A lower bound on Voronoi diagram complexity. *Information Processing Letters* 83 (2002), 183–185.

[4] F. Aurenhammer and H. Imai. Geometric relations among Voronoi diagrams. *Geometriae Dedicata* 27 (1988), 65–75.

[5] F. Aurenhammer, R, Klein, and D.T. Lee. *Voronoi diagrams and Delaunay Triangulations*. World Scientific, Singapore, 2013.

[6] L.P. Chew, K. Kedem, M. Sharir, B. Tagansky, and E. Welzl. Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995, 197–204.

[7] I. Emiris, T. Malamatos, and E. Tsigaridas. Approximate nearest neighbor queries among parallel segments. *Proc. 26th European Workshop on Computational Geometry*, 2010.

[8] H. Everett, D. Lazard, S. Lazard, and M. Safey El Din. The Voronoi diagram of three lines in R^3. *Proc. 23rd Ann. Symposium on Computational Geometry*, 2007, 255–264.

[9] V. Koltun and M. Sharir. Polyhedral Voronoi diagrams of polyhedra in three dimensions. *Discrete & Computational Geometry* 31 (2002), 83–124.

[10] V. Koltun and M. Sharir. Three-dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. *SIAM Journal on Computing* 32 (2003), 616–642.

[11] J. Schwartz and M. Sharir. On the two-dimensional Davenport-Schinzel problem. *Journal of Symbolic Computation* 10 (1990), 371–393.

[12] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry* 12 (1994), 327–345.

Additive Weights for Straight Skeletons

Martin Held* Peter Palfrader*

Abstract

We introduce an *additively-weighted straight skeleton* as a new generalization of straight skeletons. It is induced by a wavefront propagation process where, unlike in standard variants, wavefront edges do not necessarily start to move at the begin of the propagation process but at later points in time.

1 Introduction

Straight skeletons were introduced to computational geometry over 20 years ago by Aichholzer et al. [2]. They have diverse applications such as in tool path generation, mathematical origami, roof design, and terrain generation. (See, for example, [7] and the references cited therein.)

The multiplicatively-weighted straight skeleton was first mentioned by Aichholzer and Aurenhammer [1] and then by Eppstein and Erickson [5]. Recently it was studied in more detail by Biedl et al. [3] who analyzed under which conditions properties of the unweighted skeleton carry over to the weighted pendant.

2 Preliminaries

A straight skeleton is defined as the outcome of a wavefront propagation process. For a simple polygon P, its wavefront $\mathcal{W}_P(t)$ changes with time t and is a set of simple polygons. Initially, at time zero, $\mathcal{W}_P(0)$ consists only of P. Then, as time increases, the edges of $\mathcal{W}_P(t)$ move towards the interior of P at unit speed in a self-parallel manner, preserving incidences. Thus, the vertices of $\mathcal{W}_P(t)$ move along the bisectors of polygon edges.

In order to maintain planarity of the wavefront during the propagation process, special processing is required to resolve non-planarities when they occur: In an *edge event*, an edge of the wavefront that has shrunk to zero length is removed. In a *split event*, a reflex vertex v reaches another part of the wavefront. The wavefront is split at this locus, and two separate polygons replace the previous polygon to restore planarity of the wavefront after the event. Typically this will happen when v reaches the interior of a wavefront edge. However, if v reaches another vertex then more complex interactions are possible [4]. The propagation process ends when all wavefront polygons have collapsed.

The traces of all vertices of $\mathcal{W}_P(t)$ over the propagation period then make up the edges of the straight skeleton $\mathcal{S}(P)$. In addition, if two parallel wavefront edges move into each other during the wavefront propagation, then the portion common to them is added to the straight skeleton while the portions that belong to only one of them remain in the wavefront [3].

To avoid ambiguities, we generally refer to the edges of the straight skeleton as *arcs* and reserve *edges* for the input polygon and the wavefront. Likewise, we call the vertices of the straight skeleton *nodes*.

The wavefront fragments of the polygon edge e at time t are contained in $\overline{e} + t \cdot n_e$, where $\overline{e}$ is the supporting line of e and n_e is its inward facing unit normal. We denote by $e(t)$ the (possibly empty) set of these wavefront fragments of edge e at time t. Every face of the straight skeleton is traced out by the fragments of exactly one input edge over time, i.e., $f(e) := \bigcup_{t \geq 0} e(t)$ for the face $f(e)$ of edge e.

Straight Skeletons with Multiplicative Weights. Multiplicatively-weighted straight skeletons, although introduced very early on, have been studied in detail only recently by Biedl et al. [3]. In the presence of multiplicative weights, wavefront edges no longer move at unit speed but instead move at different speeds depending on a weight function $\sigma\colon E \to \mathbb{R}$ where E is the edge-set of P. The wavefront fragments of the line e are contained in $\overline{e} + t \cdot \sigma(e) \cdot n_e$.

If all weights are required to be positive, then most of the well-known properties of straight skeletons are preserved. One prominent exception is that a face need not be monotone to its defining input edge any more. For negative weights, $\mathcal{S}(P)$ need not even be a tree and may contain crossings [3].

Please visit Held's CGA Lab [6] for references to prior work on straight skeletons and several examples.

3 Additively-Weighted Straight Skeletons

3.1 Definition

Given a simple polygon P and an additive-weight function $\delta\colon E \to \mathbb{R}_0^+$, we define the additively-weighted wavefront $\mathcal{W}_{P,\delta}(t)$ as follows. As in the unweighted case, $\mathcal{W}_{P,\delta}(0)$ is identical to P. However,

*Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria; supported by Austrian Science Fund (FWF) Grant P25816-N15; {held,palfrader}@cosy.sbg.ac.at.

wavefront edges do not all start to move immediately. Rather, an edge of the wavefront that is emanated from polygon edge e will only start to move inwards at unit speed at time $\delta(e)$.

Since wavefront edges no longer move all at once, wavefront vertices will not travel exclusively along bisectors of input edges. If both incident wavefront edges have not yet started to move, then the wavefront vertex will obviously remain stationary. If exactly one incident wavefront edge has started to move, then the wavefront vertex will travel on the supporting line of the other; see Figure 2.

During its propagation process the wavefront will see instances of edge and split events, and it needs to be updated accordingly to restore planarity after the event. Note that even edges and vertices that have not yet started to move can be involved in both types of events. (See for instance the edge in the top right of Figure 1, which collapses before it starts moving.) The wavefront propagation process ends when all wavefront polygons have collapsed.

The *additively-weighted straight skeleton* $S(P, \delta)$ is then defined as the geometric graph whose edges are the traces of vertices of $\mathcal{W}_{P,\delta}(t)$ over its propagation period.

As in the unweighted case, we call edges of $S(P, \delta)$ *arcs* and its vertices *nodes*. Similarly, we again call the loci traced out by the wavefront segments $e(t)$ of edge e the *face* $f(e)$ *of* e, defined as $f(e) := \bigcup_{t \geq 0} e(t)$. We call the instance when an edge starts to move a *speed-change event*.

3.2 Properties

Node Degrees. In unweighted or multiplicatively-weighted straight skeletons, a node will be of degree one when it is a leaf of the straight skeleton (its locus will then be at a vertex of the input polygon) or of degree three when it is the result of an elementary edge or split event. Higher node degrees are also possible and are induced by non-elementary events where more than three wavefront edges are involved [4].

In addition to these types of nodes, the additively-weighted straight skeleton can have nodes of degree two. These occur when a vertex of the wavefront changes its speed due to an incident wavefront edge starting to move.

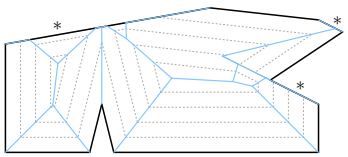


Figure 1: Polygon(black) with an additively-weighted straight skeleton (blue). The edges marked by $*$ have non-zero additive weights. A family of offset curves is shown in gray and dotted.

Nested Wavefronts. As in the case of unweighted or multiplicatively weighted straight skeletons with non-negative weights, the wavefronts of additively weighted straight skeletons are nested inside each other.

Lemma 1 *Let $t_1, t_2 \in \mathbb{R}_0^+$ with $t_1 < t_2$. Then $\mathcal{W}_{P,\delta}(t_2)$ lies within $\mathcal{W}_{P,\delta}(t_1)$.*

Proof. Edges that are already moving at time t_1 keep moving towards the interior of $\mathcal{W}_{P,\delta}(t_1)$. Edges that are still stationary do not move towards the outside either: Their incident vertices may move, however, as long as they stay on the same supporting line. As illustrated in Figure 2, these vertices do not move to the outside of the wavefront polygon either. □

Crossings, Planarity, and Connectedness. At time zero, the wavefront $\mathcal{W}_{P,\delta}(0)$ is a single simple polygon. Let t_1 be the earliest time at which an edge starts to move that was not moving initially, i.e., the time of the first non-trivial speed-change event. Then in the time interval $[0, t_1]$ the wavefront will propagate the same as if it were the wavefront of a multiplicatively-weighted polygon, where the weight of each edge is either 1 if it is already moving, or 0 if it is not yet moving.

Biedl et al. [3] show that the multiplicatively-weighted straight skeleton is free of crossings[1] for positive edge weights σ. This result extends to weights including zero. Therefore, the wavefront propagation in the time interval $[0, t_1]$ will not trace out any wavefront arcs that cross other arcs.

Now suppose that the wavefront propagation has not ended by time t_1. Then at time t_1 one or more edges will start to move and the wavefront $\mathcal{W}_{P,\delta}(t_1)$ will consist of one or more polygons. Lemma 1 implies that we can apply our reasoning to each wavefront polygon individually. If t_2 is the time of the next speed-change event, then by the same argument no wavefront arcs traced out during $[t_1, t_2]$ will cross either. Furthermore, arcs traced out during $[t_0, t_1]$ and $[t_1, t_2]$ will be confined to the areas that the wavefront traced out during their respective time. So no arc traced out during the latter interval can cross an arc traced out during the former. By induction, this claim holds for the entire wavefront propagation process.

Lemma 2 *The additively-weighted straight skeleton of a simple polygon is free of crossings.*

Note however, that we cannot infer strict planarity from being free of crossings: Assume v is a wavefront

[1] Roughly, a geometric graph G contains a crossing if there exists an arbitrarily small disk B centered on the interior of an arc such that no (open) half of B is empty of elements of the graph, or if two nodes of G share the same locus.

vertex where one incident edge e has not yet started to move. Let e' be the other edge incident at v. Then v travels on the supporting line of e, and the direction of this movement depends on the angle that e spans with e'; see Figure 2.

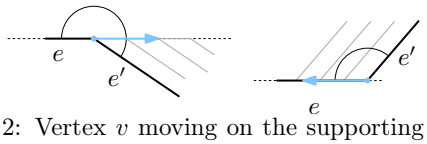


Figure 2: Vertex v moving on the supporting line of wavefront edge e that has not started to move yet.

Now let e' collapse in an edge event. Let e'' be the new neighbor of e and let v' be the new vertex that was created in this edge event. (At the time of the event, v' will be in the same locus as v, which it replaces in the wavefront.) If the angle at v' is now convex where at v it previously was reflex, then v' will move in the opposite direction of v. This results in the arc being traced out by v' to overlap the arc already traced out by v; see Figure 3.

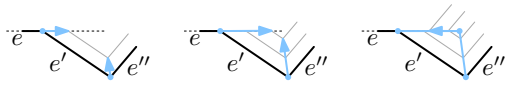


Figure 3: After an edge event, a wavefront vertex may backtrack along an arc previously traced out.

Lemma 3 *Let $\mathcal{S}_{P,\delta}(t)$ be the straight-skeleton features traced by the wavefront until time t. If two points $p, q \in \mathcal{S}_{P,\delta}(t)$ are path-connected on $\mathcal{S}_{P,\delta}(t) \cup \mathcal{W}_{P,\delta}(t)$, then they are path-connected on $\mathcal{S}(P,\delta)$.*

Proof. This is shown for multiplicatively-weighted straight skeletons in Lemma 13 of [3] by induction on the events of the wavefront propagation in chronological order. We know from their proof that connectivity of p and q is not broken by edge and split events. Therefore, it only remains to show that connectivity is maintained across speed-change events.

First, a speed-change event does not change the combinatorial properties of the wavefront. It will only result in vertices potentially moving at different speeds. Thus, a path between p and q cannot be split by changes to the wavefront.

Second, let us consider arcs being traced out by a vertex v of the wavefront. If v is not incident to a wavefront edge affected by a speed-change event at time t, it will just continue across the event with no change, tracing out a continuous arc. If, however, v is incident to an affected edge, then it will change direction, and therefore the arc that it traced out up until t will end at the position of v at time t. However, there it will connect via a node of degree two to the arc that v is tracing out after t. Thus, a path that goes over an arc currently being traced out by the wavefront cannot be disconnected by a speed-change event. □

Corollary 4 *The additively-weighted straight skeleton of a simple polygon is connected.*

Faces. For each edge e of P, we defined its face as $f(e) := \bigcup_{t\geq 0} e(t)$, where $e(t)$ is the set of segments of the wavefront at time t that were emanated by e. Initially, $e(0)$ will consist of only one segment that coincides with e, but as the wavefront propagates, segments may get split and segments may get dropped from $e(t)$ when they collapse. However, at no time will a segment just jump into existence. Hence each face is connected.

Note, however, that $f(e)$ is not necessarily a simple polygon for edges that do not immediately start to move. The faces in Figure 1 that correspond to the edges with non-zero additive weights demonstrate this fact. In clockwise order from the top left, we have a face whose interior is disconnected, a face with an empty interior because its corresponding edge collapsed before it started to move, and a face whose interior is not adjacent to e itself.

In the unweighted straight skeleton, the face of an edge e is a monotone polygon with respect to the supporting line of e. This is however not always the case for additively-weighted straight skeletons; see for example the topmost face in Figure 1.

Lemma 5 *A face of an additively-weighted straight skeleton need not be monotone with respect to the edge that emanated it.*

Roof Model. The roof model [2] raises the wavefront propagation into three-space, with the (third) z-coordinate being the time t. With P embedded in the $t = 0$ plane, the wavefronts over time thus form a polytope over P. This piecewise linear and continuous polytope $R(P) := \bigcup_{t\geq 0}(\mathcal{W}_P(t) \times \{t\})$ is called the *roof* of P. For unweighted straight skeletons the roof is a terrain (z-monotone).

The roof model is a useful theoretical tool when dealing with straight skeletons as it makes some proofs easier. It is also directly useful as a solution for modeling terrains or actual roofs of buildings.

The roof in the additively-weighted case is defined similarly as $R(P,\delta) := \bigcup_{t\geq 0}(\mathcal{W}_{P,\delta}(t) \times \{t\})$. It clearly is no longer strictly z-monotone, since wavefront edges may stay on the same supporting line during the propagation, resulting in vertical facets. The house depicted in Figure 4 has many such facets, namely the walls, as all input edges have (different) additive weights assigned to them. The weight assigned to some edges is larger, resulting in some walls continuing upwards while inclined roof facets already exist at the same height.

Lemma 6 *The roof $R(P,\delta)$ induced by an*

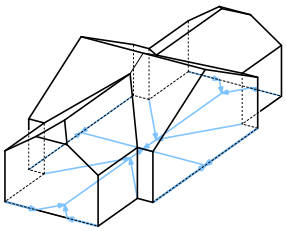


Figure 4: A house with a roof induced by an additively-weighted straight skeleton.

additively-weighted straight skeleton is weakly z-monotone.

Proof. This is a direct consequence of Lemma 1. □

For each edge e of the polygon, the roof will have at least one facet, the one incident to e. If the segments of e see a speed-change event during the propagation process, one additional facet per segment will be visible in the roof. Note that all facets of e correspond to only a single face in the straight skeleton as the "bend" caused by the speed-change event is not apparent in $\mathcal{S}(P, \delta)$. Also note that the total number of facets is still linear in the size of the input polygon since additional wavefront segments can only be caused by split events which are bounded linearly in the input size.

4 Generalizations

Several generalizations seem natural. First, one can combine multiplicative weights with additive weights. We postulate that no properties change as long as the multiplicative weights remain non-negative. If negative additive weights are allowed then the wavefront propagation would simply start at a time corresponding to the smallest negative weight. In terms of the roof model, negative weights merely mean shifting the whole structure along the z-axis.

Second, the additively-weighted straight skeleton can of course be defined not only for simple polygons but for planar straight-line graphs also.

Third, one could even allow more than a single speed-change per edge. As long as the speed function for an edge remains piecewise constant, vertices would still move along straight lines and the straight skeleton would be quite recognizable.

5 Computation

A simple method is described by Aichholzer et al. [2] to compute the unweighted straight skeleton: Compute the $\mathcal{O}(n)$ many collapse times of all edges and the $\mathcal{O}(n^2)$ times of all potential split events, and maintain them in a priority queue. On events, only a constant number of edge collapses have to be recomputed at constant cost each. Since the total number of events is linear, the overall algorithm runs in $\mathcal{O}(n^2 \log n)$ time at the cost of $\mathcal{O}(n^2)$ memory, where n is the size of the input polygon.

For the additively-weighted straight skeleton, the same approach can be used. The computation of potential split event times is slightly more involved but still in $\mathcal{O}(n^2)$. On speed-change events, a possibly linear number of collapses have to be recomputed, but the amortized cost for these is still only linear. Therefore, the additively-weighted straight skeleton can also be computed in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space.

As in our prior work [8], we use a variant of Aichholzer and Aurenhammer's triangulation-based algorithm [1] for our implementation. Their core idea is to maintain a kinetic triangulation of the wavefront polygons, and keep track of triangle collapses in a priority queue as these signal events. We augmented the priority queue with the times of speed-change events, and are thus able to compute the additively-weighted straight skeleton.

Our implementation is based on CGAL and is capable of exactly computing the straight skeleton of a planar straight-line graph with non-negative additive and multiplicative weights. For instance, the straight skeletons and offsets in Figures 1 and 4 were produced by our code.

References

[1] O. Aichholzer and F. Aurenhammer. Straight Skeletons for General Polygonal Figures in the Plane. In *Voronoi's Impact on Modern Sciences II*, volume 21, pages 7–21. Institute of Mathematics of the National Academy of Sciences of Ukraine, 1998.

[2] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A Novel Type of Skeleton for Polygons. *J. Univ. Comp. Sci*, 1(12):752–761, 1995.

[3] T. Biedl, M. Held, S. Huber, D. Kaaser, and P. Palfrader. Weighted Straight Skeletons in the Plane. *Comp. Geom.: Theory and Appl.*, 48(2):120–133, Feb. 2015.

[4] T. Biedl, S. Huber, and P. Palfrader. Planar Matchings for Weighted Straight Skeletons. In *Proc. ISAAC*, pages 117–127, 2014.

[5] D. Eppstein and J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete & Comp. Geom.*, 22(4):569–592, 1999.

[6] M. Held. Weighted & Unweighted Straight Skeletons. `https://www.cosy.sbg.ac.at/~held/projects/wsk`.

[7] S. Huber. *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*. Shaker Verlag, 2012.

[8] P. Palfrader, M. Held, and S. Huber. On Computing Straight Skeletons by Means of Kinetic Triangulations. In *Proc. ESA*, pages 766–777, 2012.

Bisector Graphs for Min-/Max-Volume Roofs over Simple Polygons

Günther Eder* Martin Held* Peter Palfrader*

Abstract

Piecewise-linear terrains ("roofs") over simple polygons were studied by Aichholzer et al. (1995) in their work on straight skeletons of polygons. We show how to construct a roof over a simple polygon that has minimum (or maximum) volume among all roofs that drain water. Such a maximum-volume (minimum-volume) roof can have quadratic (maybe cubic, resp.) number of facets. Our algorithm for computing such a roof extends the standard wavefront propagation known from the theory of straight skeletons by two additional events. Both the minimum-volume and the maximum-volume roof of a simple polygon with n vertices can be computed in $\mathcal{O}(n^3 \log n)$ time.

1 Introduction

1.1 Motivation and Prior Work

In 1995 Aichholzer et al. [3] introduced straight skeletons of simple polygons. Their work also highlights the intimate connection between straight skeletons — as a special form of a bisector graph — of polygons in the two-dimensional plane and a 3D structure called "roof". The bisector graph and the roof model are used to demonstrate straight skeleton properties. They mention that its roof volume is neither maximized nor minimized. Their algorithm uses a sweep-plane approach to compute the straight skeleton of a simple polygon in $\mathcal{O}(n^2 \log n)$ time. Aichholzer and Aurenhammer [2] apply a wavefront propagation to compute straight skeletons of general planar straight-line graphs.

While every straight skeleton of a simple polygon has its corresponding roof [3], it seems natural to study also other types of roofs. Indeed, so-called "realistic roofs" were introduced in recent work by several authors [6, 1]. Their approach enumerates all possible realistic roofs over a rectilinear polygon in $\mathcal{O}(n^5)$ time. A side result of their work is the computation of a realistic roof that has minimum height or minimum volume (under the roof).

We pick up this lead and generalize realistic roofs to "natural roofs": Roughly, we still require a natural roof to drain water but wave the restriction that every facet of the roof has to be connected to its defining boundary edge. (See the gray triangular area in Fig. 1b.) We show how to employ a wavefront propagation to compute a minimum-volume (maximum-volume) roof of a simple polygon with n vertices in $\mathcal{O}(n^3 \log n)$ time.

*Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria; Work supported by Austrian Science Fund (FWF) Grant P25816-N15; {geder,held,palfrader}@cosy.sbg.ac.at.

1.2 Basics

Throughout this paper we let P denote a simple polygon in the xy-plane, Π_0, of $\mathbb{R}^3$. The *interior side* of an edge e of P is the half-plane (within Π_0) induced by its supporting line $\ell(e)$ which locally (close to e) overlaps with the interior of P. We associate a half-plane $\Pi(e)$ with e in the following way: (i) The intersection of $\Pi(e)$ with Π_0 is given by $\ell(e)$, (ii) $\Pi(e)$ lies within the half-space $z \geq 0$ of $\mathbb{R}^3$, (iii) the normal projection of $\Pi(e)$ onto Π_0 coincides with the interior side of e; i.e., $\Pi(e)$ is inclined towards the interior of P, and (iv) $\Pi(e)$ forms a 45° angle with Π_0.

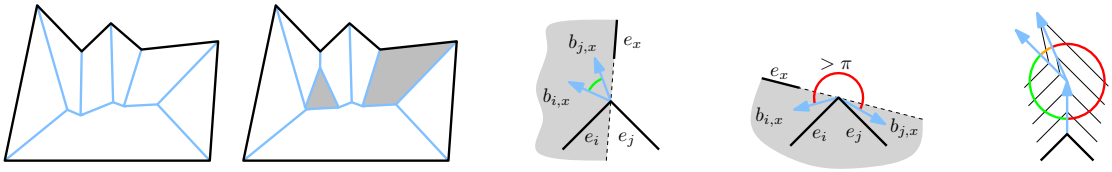
Consider two different edges e_1, e_2 of P. The (angular) *bisector* of e_1, e_2 is the set of all points within the intersection of the interior sides of e_1 and e_2 that are equidistant from $\ell(e_1)$ and $\ell(e_2)$.

For the sake of (mostly descriptional) simplicity we assume that P is in general position: (i) No two edges of P are parallel to each other, and (ii) not more than three bisectors of edges of P meet in one point. Under this assumption, the bisector of two edges e_1, e_2 of P is a ray that starts at the point of intersection $\ell(e_1) \cap \ell(e_2)$ and leads into the common interior of e_1 and e_2.

Definition 1 (Bisector Graph [3]) *A connected planar straight-line graph is a* bisector graph, $\mathcal{B}(P)$, *of P if (i), all its edges are portions of bisectors of edges of P, (ii) it has no degree-two node, and (iii) there is a bijection between its degree-one nodes and the vertices of P.*

The straight skeleton of P is known to be one specific bisector graph of P [3]; cf. Fig. 1a and 1b. The edges of a bisector graph are called $\mathcal{B}$-arcs, and common end-points of $\mathcal{B}$-arcs are called $\mathcal{B}$-nodes. By letting a $\mathcal{B}$-arc of $\mathcal{B}(P)$ inherit the orientation of its supporting bisector ray we impose an orientation onto every $\mathcal{B}$-arc, thus turning a bisector graph into a *directed bisector graph*. Naturally, $\mathcal{B}$-nodes in a directed bisector graph have an in-degree and out-degree.

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.



(a) (b) (c) (d) (e)

Figure 1: In (a), we see the straight skeleton of a simple polygon P, in (b) we see another bisector graph of P. In (c), we see a create event while in (d), due to an internal angle greater than π (red), no create event is given. In (e), we see how a create event modifies the wavefront polygon.

Definition 2 (Roof Model [3]) *A* roof *for P, $\mathcal{R}(P)$, is a terrain over P, i.e., the graph of a piecewise-linear continuous function over P, such that (i) every facet of $\mathcal{R}(P)$ is a maximal connected subset of a half-plane $\Pi(e)$ of some edge e of P, and (ii) the intersection of $\mathcal{R}(P)$ with Π_0 is equal to the boundary of P.*

Theorem 1 ([3]) *Every roof for P corresponds to a unique bisector graph of P, and vice versa.*

We say that an *edge e* of P *defines a facet f* of $\mathcal{R}(P)$ if f is contained in $\Pi(e)$. Note that some edge may define multiple facets. As usual, a vertex v of P is called reflex if the internal angle at v is greater than π; convex otherwise. We call an edge e between two neighboring facets of a roof *valley* or *ridge* depending on whether e originates from a reflex or convex vertex.

Consider a facet f of a roof of P, and let f' be its normal projection onto Π_0. The *truncated prism defined by f* is the solid bounded by f, f' and by trapezoids between all pairs of corresponding edges of f and f'.

If all facets of the roof of a house have the so-called gradient property then water is guaranteed to drain and local minima are omitted [3]. Since the gradient property requires every point on a facet of an edge e to have a steepest path to e this is a sufficient but not a necessary condition condition for water to drain. We consider a less stringent requirement for a roof to drain water and still omit local minima.

Definition 3 (Natural Gradient Property) *Let $\mathcal{R}(P)$ be a roof for P. We say that a facet f of $\mathcal{R}(P)$ has the* natural gradient property *(*NGP*) if, for every point $p \in f$, there exists a path $g(p)$ that (i) starts at p, (ii) follows the steepest gradient, and (iii) reaches the boundary of P.*

Definition 4 (Natural Roof) *A roof $\mathcal{R}(P)$ for a polygon P is called a* natural roof for P *if all its facets have the natural gradient property.*

Definition 5 (Min-/Max-Vol. Bisector Graph) *The* maximum-volume bisector graph $\mathcal{B}_{\max}(P)$ *of a polygon P is a bisector graph $\mathcal{B}(P)$ whose associated roof $\mathcal{R}(P)$ is a natural roof that maximizes the volume over all possible natural roofs for P. Similarly for the* minimum-volume bisector graph $\mathcal{B}_{\min}(P)$.

Definition 6 (Capped Roof) *A* capped (natural) roof with height $t \geq 0$ *for a polygon P is the set of all points of a (natural) roof $\mathcal{R}(P)$ of P whose z-coordinate does not exceed t.*

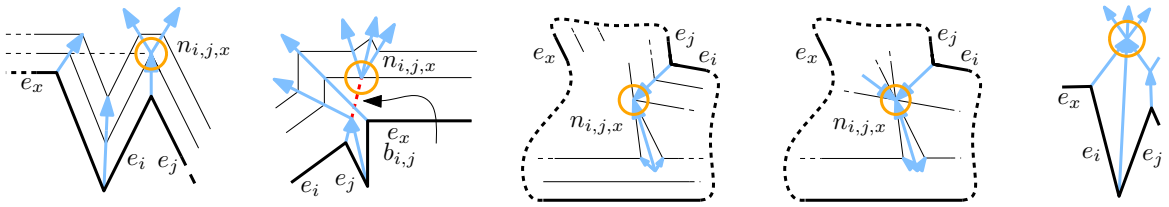
2 Computing Min-/Max-Volume Roofs

Wavefront propagation [3, 2] is a well-known strategy for computing straight skeletons. Roughly, the wavefront propagation of P is a shrinking process in which every input edge of P is offset inwards in a self-parallel manner. Initially, the segments of the wavefront correspond to the edges of the polygon. During the wavefront propagation every wavefront segment moves at unit speed towards the interior of P. It is common to regard the wavefront as a function of time t and to write $\mathcal{W}_P(t)$ to denote the shrinking (wavefront) polygons at time t. At time t every wavefront segment is at normal distance t from its input edge. As time progresses, the normal distance of each wavefront segment to its defining input edge grows. The points of intersection between consecutive wavefront segments lie on the bisectors of their defining input edges.

The wavefront vertices move along these bisectors and trace out the desired bisector graph. In order to maintain the planarity of $\mathcal{W}_P(t)$, and to obtain a straight skeleton, one has to handle two events: An *edge event* occurs when a wavefront segment shrinks to zero length. A *split event* occurs when a wavefront vertex crashes into a wavefront edge which moves in the opposite direction. A split event results in the split of the wavefront polygon into two sub-polygons.

Observation 1 (Vertex Speed [4]) *The speed of a wavefront vertex v is given by $s(v) = \frac{1}{\sin(\alpha/2)}$, where α is the exterior angle of v.*

We will also employ a wavefront propagation to compute a min-/max-volume bisector graph. Our process involves four event types: *edge event*, *split event*, as in the straight skeleton computation [3], and



(a) (b) (c) (d) (e)

Figure 2: In (a–b), we see create events and in (c–d) we see divide events; and (e) shows a bisector graph that is not produced by our propagation scheme. Note that the (red) dashed line in (b) is part of the bisector $b_{i,j}$ on which a stealth vertex moves.

divide event and *create event* as two new event types. In addition, we employ so-called *stealth vertices*.

Definition 7 (Stealth Vertex) *Let $p_{i,j} := \ell(e_i) \cap \ell(e_j)$ be the point of intersection of the supporting lines of two edges e_i, e_j of P. As the wavefront propagates, $p_{i,j}$ moves inwards along the bisector $b_{i,j}$ of e_i, e_j, at a speed given by Observation 1. At any time when $p_{i,j}$ is not part of the wavefront polygon we call it a* stealth vertex *of P. (See Fig. 2b.)*

Definition 8 (Create Event) *If (1) the supporting line of a wavefront edge e becomes incident with a reflex wavefront vertex v, where e is not incident at v, or (2) a stealth vertex v becomes incident with a wavefront edge e, and for either (1) or (2) the interior angle between the two bisectors between e and the two edges incident at v is smaller than π, then we call it a* create event. *(See Fig. 1c and 1d.)*

In the former case we insert an edge with zero length between the two edges defining the reflex vertex. The new edge belongs to the same input edge as the one defining the supporting line; cf. Fig. 2a. In the latter case we insert two edges with zero length at the point of intersection, thus splitting the intersected wavefront edge. The two edges associated with the stealth vertex define the two new edges and the stealth vertex becomes a wavefront vertex; cf. Fig. 2b.

We recall that the general position of P prohibits parallel input segments. However, according to its definition a create event adds parallel wavefront segments to a wavefront polygon. This is a necessary condition for the *divide event*.

Definition 9 (Divide Event) *When two or three reflex wavefront vertices become incident and all incident wavefront edges originate from three common input edges we call it a* divide event.

In the former case the two parallel edges (associated with one input edge) join into one edge and the two remaining edges become adjacent; cf. Fig. 2c. In the latter case three parallel edge pairs become incident and all edges change adjacencies to their neighbor; cf. Fig. 2d.

We point out that the divide event is not a "vertex-event" [5] in disguise where reflex wavefront vertices become incident as well. Note that all events but the create event are compulsory: Ignoring only one of them during the wavefront propagation would result in a self-intersecting wavefront. Only create events are optional: Accepting or ignoring such an event gives us the freedom to construct different roofs and, thus, to influence the volume of the resulting roof.

Every event takes place at the intersection of three bisectors and forms a $\mathcal{B}$-node in the bisector graph. Three $\mathcal{B}$-arcs start or end at every $\mathcal{B}$-node, except for three cases: (i) at the vertices of P; (ii) degree-six nodes that occur in a divide event where three reflex wavefront vertices become incident; cf. Fig. 2d; and (iii) another degree-six node which is not listed as an event; cf. Fig. 2e. However, in (i) no event takes place and in (ii) the in-degrees and the out-degrees both are three and the directions of the incident $\mathcal{B}$-arcs alternate when one moves around the node. Lastly, (iii) is not relevant in our propagation schema as it can be omitted due to the goal of min-/maximizing.

Besides these exceptions, no two $\mathcal{B}$-arcs can lie on a common bisector and intersect at a common $\mathcal{B}$-node. In the full paper we list all combinatorically possible bisector embeddings for a $\mathcal{B}$-node and show that every combination is considered.

By definition, all events occur at intersections of bisectors of P. Furthermore, before and after each event all wavefront vertices advance on bisectors of P. The proofs of the subsequent claims are contained in the full paper.

Lemma 2 *Any wavefront propagation results in a bisector graph.*

Lemma 3 *Any wavefront propagation results in a roof.*

Lemma 4 *A capped roof of height t, constructed using a wavefront propagation, fulfills the natural gradient property for all its facets.*

Lemma 5 *A bisector graph can be seen as a directed acyclic graph (DAG).*

Lemma 6 *The speed of a wavefront vertex v defines the slope of its associated ridge or valley, with respect to v and Π_0.*

Corollary 7 *A slower wavefront vertex leads to a locally larger slope of its associated ridge or valley, and vice versa.*

Lemma 8 *The volume of a natural roof created by a wavefront propagation can only be influenced by a create event.*

Lemma 9 *A create event takes place at a reflex wavefront vertex p. A small disc c centered at p is partitioned into three wedges by the three $\mathcal{B}$-arcs incident at p. If one wedge has an angle greater than π it involves a wavefront vertex, starting at p, that moves faster than the wavefront vertex which ends at p; cf. Fig. 1e.*

Definition 10 (De-/Accelerating Create Event) *Accepting a create event during the wavefront propagation results in new wavefront vertices. If one of the new wavefront vertices moves faster than the intersected wavefront edge or vertex then we call it an "accelerating" create event. If all new wavefront vertices move slower than the intersected wavefront vertex then we call it a "decelerating" create event.*

Lemma 10 *A create event with out-degree three is always an accelerating create event.*

Note that this implies that decelerating create events can only occur on reflex wavefront vertices, i.e., for the first case of Definition 8(1). Furthermore, Lemma 9 implies that Definition 10 is complete; there is no third class of create events.

Lemma 11 *A decelerating (accelerating) create event increases (decreases, resp.) the roof volume.*

Summarizing, a faster moving vertex increases the area that is swept by the wavefront, thus resulting in a locally reduced roof volume. Other propagation events that occur earlier can, at most, lead to a reduction of the roof volume. Conversely for a slower moving vertex. Hence, one run of the wavefront propagation (without backtracking) suffices to obtain a minimum-volume or maximum-volume roof. We summarize our result in the following theorem.

Theorem 12 *Accepting all decelerating (accelerating) create events during the wavefront propagation leads to $\mathcal{B}_{\max}$ ($\mathcal{B}_{\min}$, respectively).*

3 Analysis

We use a wavefront propagation for both $\mathcal{B}_{\min}(P)$ and $\mathcal{B}_{\max}(P)$. The complexity is dominated by the computation of all create events. For $\mathcal{B}_{\max}(P)$, one reflex input vertex can result in $\mathcal{O}(n)$ create events. Thus, we can get a quadratic number of facets. To handle one create event during the propagation takes $\mathcal{O}(n \log n)$ time. The overall complexity is therefore $\mathcal{O}(n^3 \log n)$ time and $\mathcal{O}(n^2)$ space. For $\mathcal{B}_{\min}(P)$, it is unclear if more than $\mathcal{O}(n^2)$ create events can occur. Hence, $\mathcal{B}_{\min}(P)$ may admit a cubic number of facets which leads to $\mathcal{O}(n^3)$ space, but the same time complexity.

4 Extensions

Our natural roofs can be regarded as a generalization of realistic roofs [6, 1], but our current definitions prevent a clean mathematical statement regarding any subset relation among these two types of roofs: The work of [6, 1] is restricted to rectilinear polygons, while we exclude parallel input edges explicitly. However, since parallel segments might occur during the wavefront propagation, we suspect that the restriction on the input edges can be waived. This would permit to apply our approach to the setting of [6, 1], thus reducing the time complexity for finding a minimum-volume realistic roof from $\mathcal{O}(n^5)$ to $\mathcal{O}(n^3 \log n)$. Obtaining a minimum-height roof is ongoing work, too.

References

[1] H.-K. Ahn, S. Bae, C. Knauer, M. Lee, C.-S. Shin, and A. Vigneron. Generating Realistic Roofs over a Rectilinear Polygon. In *Algorithms and Computation*, volume 7074, pages 60–69. 2011.

[2] O. Aichholzer and F. Aurenhammer. Straight Skeletons for General Polygonal Figures in the Plane. In *Proc. 2nd International Computing and Combinatorics Conference (COCOON 1996)*, volume 1090, pages 117–126, Hong Kong, 1996.

[3] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. Straight Skeletons of Simple Polygons. In *Proc. 4th International Symposium of LIESMARS*, Wuhan, China, 1995.

[4] S.-W. Cheng and A. Vigneron. Motorcycle Graphs and Straight Skeletons. In *Proc. 13th Symposium on Discrete Algorithms (SODA 2002)*, pages 156–165, San Francisco, CA, USA, 2002.

[5] D. Eppstein and J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete and Computatational Geometry*, 22(4):569–592, 1999.

[6] J. Sherette and S. Yoon. Realistic Roofs over a Rectilinear Polygon Revisited. In *Computing and Combinatorics*, pages 233–244. Springer Berlin Heidelberg, 2013.

Stabbing circles for some sets of Delaunay segments

Mercè Claverol[†] Elena Khramtcova[‡] Evanthia Papadopoulou[‡] Maria Saumell[§] Carlos Seara[†]

Abstract

Let S be a set of n disjoint segments in the plane that correspond to edges of the Delaunay triangulation of some fixed point set. Our goal is to compute all the combinatorially different stabbing circles for S, and the ones with maximum and minimum radius. We exploit a recent result to solve this problem in $O(n \log n)$ time in two cases: (i) all segments in S are parallel; (ii) all segments in S have the same length. We also show that the problem of computing the stabbing circle of minimum radius of a set of n parallel segments of equal length (not necessarily edges of a Delaunay triangulation) has an $\Omega(n \log n)$ lower bound.

1 Introduction

The *stabbing circle problem* is formulated as follows: Let S be a set of n segments in $\mathbb{R}^2$ in general position (segments have $2n$ distinct endpoints, no three endpoints are collinear, and no four of them are cocircular). A circle c is a stabbing circle for S if exactly one endpoint of each segment of S is contained in the exterior of the closed disk induced by c; see Fig. 1. The stabbing circle problem consists of (1) reporting a representation of all the combinatorially different stabbing circles for S (two circles are *combinatorially different* if the sets of endpoints in the exterior of the corresponding disks are different); and (2) finding stabbing circles with minimum and maximum radius.

Figure 1: Left: Segment set with a stabbing circle. Right: Segment set with no stabbing circle.

The stabbing circle problem has antecedents in the stabbing line problem, which was solved in optimal $\Theta(n \log n)$ time by Edelsbrunner et al. [6]. Other stabbing shapes (wedges, isothetic rectangles, etc.) have also been considered; see [4] for an overview.

The problem of stabbing a set S of n segments in the plane by a circle can be solved in $O(n^2)$ time by a combination of known results, and this is worst-case optimal [4]. Recently, we presented an alternative algorithm based on connecting the problem to *cluster Voronoi diagrams* [4]. We identified conditions under which the algorithm is subquadratic; these conditions are: (1) the Hausdorff Voronoi diagram and the farthest-color Voronoi diagram have linear structural complexity and can be constructed in subquadratic time (see Section 2 for the definition of these diagrams); (2) a technical condition related to the number of times an edge of the Hausdorff Voronoi diagram contains centers of combinatorially different stabbing circles. If the segments in S are parallel, conditions (1) and (2) are satisfied, and the stabbing circle problem for S can be solved in $O(n \log^2 n)$ time.

In this note we continue investigating special instances of segment sets for which the algorithm in [4] is subquadratic, in order to understand the stabbing circle problem better. We focus on sets S of disjoint segments that correspond to edges in the Delaunay triangulation of a fixed point set. We solve the stabbing circle problem in $O(n \log n)$ time when all segments in S are either parallel or have the same length. We also show an $\Omega(n \log n)$ lower bound for the problem of computing the stabbing circle of minimum radius of a set of n parallel segments of equal length (not necessarily edges of a Delaunay triangulation).

2 Preliminaries

In what follows, xx' denotes either a segment in S, or the pair of its endpoints as convenient.

Definition 1 [5, 9] *The* Hausdorff Voronoi diagram *of S is a partitioning of $\mathbb{R}^2$ into the following regions:*

$$\mathsf{hreg}(aa') = \{p \in \mathbb{R}^2 \mid \forall bb' \in S \setminus \{aa'\}: \max\{d(p,a), d(p,a')\} < \max\{d(p,b), d(p,b')\}\};$$
$$\mathsf{hreg}(a) = \{p \in \mathsf{hreg}(aa') \mid d(p,a) > d(p,a')\}.$$

The *graph structure* of this diagram is $\mathsf{HVD}(S) = \mathbb{R}^2 \setminus \bigcup_{aa' \in S} (\mathsf{hreg}(a) \cup \mathsf{hreg}(a'))$. An edge of $\mathsf{HVD}(S)$ is *pure* if it is incident to regions of two distinct segments.

[†]Universitat Politècnica de Catalunya, Spain. E-mails: {merce.claverol,carlos.seara}@upc.edu. Supported by projects MINECO MTM2015-63791-R and Gen.Cat. DGR2014SGR46

[‡]Faculty of Informatics, Università della Svizzera italiana (USI), Switzerland. E-mails: {elena.khramtcova, evanthia.papadopoulou}@usi.ch. Supported by SNF project 20GG21-134355, under the ESF EUROCORES, EuroGIGA/VORONOI program.

[§]Department of Mathematics and European Centre of Excellence NTIS, University of West Bohemia, Czech Republic. E-mail: saumell@kma.zcu.cz. Supported by project LO1506 of the Czech Ministry of Education, Youth and Sports.

Definition 2 [1, 7] *The* farthest-color Voronoi diagram *is a partitioning of* $\mathbb{R}^2$ *into the following regions:*

$$\mathsf{fcreg}(aa') = \{p \in \mathbb{R}^2 \mid \forall bb' \in S \setminus \{aa'\}: \min\{d(p,a), d(p,a')\} > \min\{d(p,b), d(p,b')\}\};$$
$$\mathsf{fcreg}(a) = \{p \in \mathsf{fcreg}(aa') \mid d(p,a) < d(p,a')\}.$$

The graph structure of this diagram is $\mathsf{FCVD}(S) = \mathbb{R}^2 \setminus \bigcup_{aa' \in S} (\mathsf{fcreg}(a) \cup \mathsf{fcreg}(a'))$.

For arbitrary segments, the combinatorial complexity of both diagrams is $O(n^2)$ [9, 1]. If the segments are disjoint, the complexity of $\mathsf{HVD}(S)$ is O(n) [5].

Let $\overline{\mathsf{hreg}}(\cdot)$ and $\overline{\mathsf{fcreg}}(\cdot)$ denote the closures of the respective Voronoi regions.

Definition 3 *Given a point* p*, the* Hausdorff disk *of* p*, denoted* $D_h(p)$*, is the closed disk centered at* p *of radius* $d(p,a)$*, where* $p \in \overline{\mathsf{hreg}}(a)$.

Let S be a set of n pairwise disjoint segments in $\mathbb{R}^2$ in general position; let S have no stabbing line. In [4] we presented an algorithm to solve the stabbing circle problem for S. To state it, we need some notation.

Let e be a pure edge of $\mathsf{HVD}(S)$ and let w be a point in e. In [4] we defined a set $type(w)$, whose elements might be $\tilde{l}$, $\tilde{r}$, mm, in, and out. The meaning of $type(w)$ is not essential for this note; it is enough to point out that $type(w)$ can be found in $O(1)$ time if w is located in $\mathsf{FCVD}(S)$.

The *find-change query* is defined as follows: Given two points t, s in e such that $type(t)$ contains $\tilde{r}$ but not $\tilde{l}$, and $type(s)$ contains $\tilde{l}$ but not $\tilde{r}$, the query returns a point w in the segment ts such that one of the following holds: (i) $\{\tilde{r}, \tilde{l}\} \subseteq type(w)$; (ii) $in \in type(w)$; (iii) $out \in type(w)$.

Suppose that $e = uv$ is a portion of the border of $\mathsf{hreg}(a)$ and $\mathsf{hreg}(b)$, for $aa', bb' \in S$. We say that a segment $cc' \in S \setminus \{aa', bb'\}$ is of type *middle* for e if either c or c' is contained in $D_h(u) \setminus D_h(v)$ and the other endpoint in $D_h(v) \setminus D_h(u)$.

Let m denote the number of pairs formed by a segment $cc' \in S$ and a pure edge e of $\mathsf{HVD}(S)$ such that cc' is of type *middle* for e. We build the results of Section 3 of this abstract on the following result.

Theorem 1 [4] *The stabbing circle problem for* S *can be solved in* $O(\mathcal{T}_{\mathsf{HVD}(S)} + \mathcal{T}_{\mathsf{FCVD}(S)} + |\mathsf{HVD}(S)|\mathcal{T}_{fc} + |\mathsf{FCVD}(S)| \log n + m\mathcal{T}_{fc})$ *time, where* $\mathcal{T}_{\mathsf{HVD}(S)}$ *(resp.,* $\mathcal{T}_{\mathsf{FCVD}(S)}$*) is the time to compute* $\mathsf{HVD}(S)$ *(resp.,* $\mathsf{FCVD}(S)$*),* $|\mathsf{HVD}(S)|$ *(resp.,* $|\mathsf{FCVD}(S)|$*) is the number of edges of* $\mathsf{HVD}(S)$ *(resp.,* $\mathsf{FCVD}(S)$*), and* $\mathcal{T}_{fc}$ *is the time to answer a find-change query.*

3 Segments with the Delaunay property

We say that S *satisfies the Delaunay property* if its segments correspond to edges of some Delaunay triangulation. Let us assume that S satisfies this property.

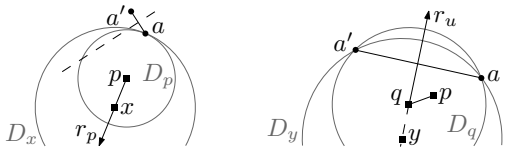


Figure 2: (a) $r_p \cap bis(a,a') = \emptyset$ and $D_p \subset D_x$. (b) $r_p \cap bis(a,a') = \{q\}$ and $D_{q\ell} \subset D_y$.

Lemma 2 $\mathsf{FCVD}(S)$ *is a tree of* $O(n)$ *complexity.*

Proof. We show that $\mathsf{FCVD}(S)$ for such a segment set S is an instance of the *farthest abstract Voronoi diagram* (FAVD); the claim then follows automatically from [8]. To prove that $\mathsf{FCVD}(S)$ is FAVD, we consider the *nearest-color* Voronoi diagram of S, which reveals the nearest site (segment in S), where the distance from a point $p \in \mathbb{R}^2$ to some $aa' \in S$ is $\min\{d(p,a), d(p,a')\}$. We need to prove that the system of bisectors for farthest/nearest color Voronoi diagram satisfies the following axioms: (1) each bisector is an unbounded Jordan curve; (2) any two bisectors intersect finite number of times; (3) regions of the nearest-color Voronoi diagram are (a) non-empty, (b) path-connected, and (c) cover $\mathbb{R}^2$. Note that the nearest-color Voronoi diagram is related to the nearest-point Voronoi diagram of all endpoints of S: the region of $aa' \in S$ in the former diagram is the union of the regions of a and a' in the latter.

Our bisector system satisfies axioms (2), (3a) and (3c) since so does the bisector system of the nearest/farthest point Voronoi diagram. Further, since each $aa' \in S$ is an edge of the Delaunay triangulation of all endpoints of S, the regions of a and a' in the nearest-point Voronoi diagram are adjacent, thus their union is path-connected, implying axiom (3b). A bisector in our system satisfies axiom (1), since it separates two unions of pairs of adjacent regions in the diagram of four points. □

The faces of $\mathsf{FCVD}(S)$ near infinity coincide with the faces of the farthest-segment Voronoi diagram of S, thus, their sequence at infinity can be computed in $O(n \log n)$ time by divide and conquer (and other methods) [10]. Based on this observation, it is simple to derive a divide and conquer algorithm for $\mathsf{FCVD}(S)$. (Note that the approach in [8] yields an expected $O(n \log n)$ time algorithm for $\mathsf{FCVD}(S)$.)

Lemma 3 $\mathsf{FCVD}(S)$ *can be constructed in* $O(n \log n)$ *time and* $O(n)$ *space.*

Let $bis(a,b)$ denote the bisector of a and b.

Lemma 4 *For a point* $p \in \mathbb{R}^2$*, let* r_p *be the open ray with origin at* p *and direction* $\overrightarrow{ap}$*, where* a *is the endpoint of* $aa' \in S$ *such that* $p \in \overline{\mathsf{fcreg}}(a)$*. Let*

$p \notin bis(a, a')$. *If* $r_p \cap bis(a, a') = \{q\}$, *then* $\mathsf{fcreg}(aa')$ *contains the open segment* pq, *as well as one of the two (unbounded) portions of* $bis(a, a')$ *starting at* q. *Otherwise,* $r_p \subset \mathsf{fcreg}(aa')$.

Proof. For any point $z \in \mathbb{R}^2$, let D_z be the disk centered at z of radius $d(z, a)$; see Fig. 2.

Suppose that r_p does not intersect $bis(a, a')$. Since $p \in \overline{\mathsf{fcreg}}(a)$, disk D_p contains an endpoint of every segment in S. For a point $x \in r_p, x \neq p$, $D_p \subset D_x$. Thus D_x contains in its interior an endpoint of every segment in $S \setminus \{aa'\}$, that is, $x \in \mathsf{fcreg}(a) \subseteq \mathsf{fcreg}(aa')$.

Suppose next that r_p intersects $bis(a, a')$ in a point q. For any point $x \in pq$, $x \neq p$, we have $x \in \mathsf{fcreg}(a) \subset \mathsf{fcreg}(aa')$ by the above argument. In particular, disk D_q contains an endpoint of every segment in S. Point q breaks $bis(a, a')$ into two rays r_u and r_ℓ, which are respectively above and below q (see Fig. 2b), and aa' breaks disk D_q into two parts D_{qu} and $D_{q\ell}$ that are above and below aa' respectively. (We assume that aa' is not vertical, otherwise the above/below relation can be replaced by left/right.) Observe that, if $\mathsf{fcreg}(aa')$ does not contain r_u (resp., r_ℓ), then $D_{q\ell}$ (resp., D_{qu}) contains an endpoint of some segment in $S \setminus \{aa'\}$. If $\mathsf{fcreg}(aa')$ contained neither r_u nor r_ℓ, there would be an endpoint of a segment in $S \setminus \{aa'\}$ inside $D_{q\ell}$, and an endpoint inside D_{qu}. A contradiction to aa' being an edge of the Delaunay triangulation of the set of endpoints of S. □

Lemma 5 $\mathsf{FCVD}(S)$ *can be preprocessed in* $O(n \log n)$ *time and* $O(n)$ *space so that a find-change query is answered in* $O(\log n)$ *time.*

Proof. By Lemma 2 $\mathsf{FCVD}(S)$ is a tree, and thus the *centroid decomposition* [3] can be built for it, and used to answer the find-change query. This decomposition is a (graph-theoretical) balanced tree with n nodes, one for each vertex of $\mathsf{FCVD}(S)$, built in $O(n \log n)$ time by finding the *centroid* vertex c of the tree $\mathsf{FCVD}(S)$, making it a root, and recursing into the three connected components of $\mathsf{FCVD}(S) \setminus \{c\}$. The subtree of each node v corresponds to a connected portion of $\mathsf{FCVD}(S)$, adjacent to the vertex v. To perform a query, we follow a root-to-leaf path (of length $O(\log n)$) in this balanced tree, at every node of the path one of the node's three subtrees is to be chosen.

We can make a decision related to one node in $O(1)$ time, thus answering a find-change query in $O(\log n)$ time. Indeed, Lemma 4 if applied to v and each of the three regions of $\mathsf{FCVD}(S)$ incident to v, induces a decomposition of $\mathbb{R}^2$ into three regions of $O(1)$ combinatorial complexity, each of which contains one subtree of v in $\mathsf{FCVD}(S)$, see Fig. 3a. Out of these three regions, in constant time we choose the only one that may contain the answer to the find-change query. □

We next bound the parameter m in Theorem 1.

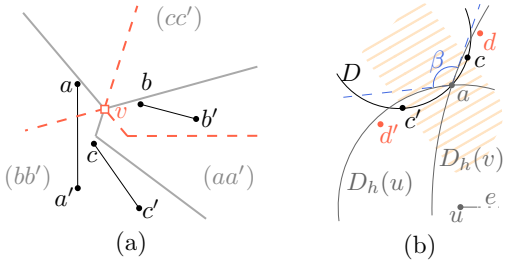


Figure 3: (a) $S = \{aa', bb', cc'\}$ (black); $\mathsf{FCVD}(S)$ (gray); the decomposition of $\mathbb{R}^2$ induced by its vertex v (red, dashed); (b) Figure for the proof of Lemma 7.

Consider a pure edge $e = uv$ of $\mathsf{HVD}(S)$ separating $\mathsf{hreg}(a)$ and $\mathsf{hreg}(b)$, for two segments $aa', bb' \in S$. Then $e \subseteq bis(a, b)$. We assume that segment ab is vertical with a on top of b, and that ab does not intersect the interior of e (otherwise e could be broken into two parts, considered separately). For any segment $cc' \in S$, we denote its supporting line by $\ell(cc')$.

Lemma 6 *If* $cc' \in S$ *is of type middle for* S, *then* $\ell(cc')$ *lies either above both* aa', bb' *or below them.*

Proof. One endpoint of cc' is in $D_h(u) \setminus D_h(v)$, and the other in $D_h(v) \setminus D_h(u)$. These two areas are separated by the vertical line $\ell(ab)$, so cc' is not vertical.

We first prove that it is impossible that a, b, c, c' are in convex position with c and c' not consecutive along the convex hull of the four points. Assume otherwise. The center of the circle through a, b and c lies on e; hence c' is outside this circle. Thus a and b are adjacent in the Delaunay triangulation of a, b, c, c'. Since this triangulation is plane, c and c' are not adjacent, and therefore they are not adjacent in the Delaunay triangulation of all endpoints of S; a contradiction.

Since c' (resp., c) is outside the circle through a, b and c (resp., c'), the convex hull of a, b, c, c' cannot be a triangle with c' (resp., c) in its interior. Hence, a and b are on the same side of $\ell(cc')$. Recall that a' and b' lie in $D_h(u) \cap D_h(v)$. Segment cc' either does not intersect $D_h(u) \cap D_h(v)$, or it divides $D_h(u) \cap D_h(v)$ in two portions, and both a, b lie in one of them. In both cases, the claim follows. □

Lemma 7 *If* S *satisfies the Delaunay property and all segments in* S *are of the same length, then an edge* e *of* $\mathsf{HVD}(S)$ *has at most two segments of type middle.*

Proof. We show that there is at most one segment of type middle whose supporting line is above aa', bb'. Then the claim follows from Lemma 6.

Suppose for contradiction that cc', dd' are segments of type middle for e such that $\ell(cc')$ and $\ell(dd')$ lie above aa', bb'. A vertical ray shot from a hits both cc' and dd'. Assume that it hits cc' first. Let D denote

the disk through c, c', a. See Fig. 3b. Since cc' is a Delaunay edge, cc' and dd' are pairwise disjoint, and a and at least one of d, d' are on opposite sides of cc', disk D contains none of d, d'.

We have $\angle dad' > \pi/2$: it is greater than the angle β formed by the two tangents to $D_h(u)$ and $D_h(v)$ at a (see blue dashed lines in Fig. 3b) and $\beta \geq \pi/2$ by our assumption that segment ab does not intersect e in its interior. Let $s(cc')$ be the closed strip formed by two lines perpendicular to $\ell(cc')$ and passing through c and c' (tiled area in Fig. 3b). We have: d, d' are outside D; d, d' are separated by $\ell(ab)$; $\angle dad' > \pi/2$; and $\ell(dd')$ lies above cc'. All this together imply that d and d' lie outside $s(cc')$ and on different sides of it. Thus $d(d, d') < d(c, c')$; a contradiction. □

Recall Theorem 1. By Lemma 5, $\mathcal{T}_{fc} = O(\log n)$. Both $\mathcal{T}_{\mathsf{FCVD}(S)}$ and $\mathcal{T}_{\mathsf{HVD}(S)}$ are $O(n \log n)$, see Lemma 3 and [4]. If all segments in S are parallel, then $m = O(n)$ [4]. By Lemma 7, m is also $O(n)$ if the segments in S have the same length. We conclude:

Theorem 8 *If S satisfies the Delaunay property and either all segments in S are parallel, or all segments in S are of equal length, then the stabbing circle problem can be solved in $O(n \log n)$ time and $O(n)$ space.*

4 Lower bound

We finally prove a lower bound for sets of segments possibly without the Delaunay property, but with the other two conditions considered in this note.

Theorem 9 *The problem of computing a stabbing circle of minimum radius for a set of n parallel segments of equal length has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model.*

Proof. The reduction, very similar to that of Theorem 6 in [2], is from $\mathsf{MAXGAP}(X)$. In our version, the input X consists of a set of n integers $x_1, \ldots, x_n$, and $\mathsf{MAXGAP}(X)$ is the problem of finding the maximum difference between consecutive elements of X.

Without loss of generality, we may assume $\min X = 1$. Let $x'_1 < x'_2 < \cdots < x'_n$ be the sorting of the elements of X. Then $x'_1 = 1$, and let $M = x'_n$. We construct a set S of parallel segments of equal length as follows: For every $x_i \in X$, we add a segment connecting point $(x_i, 0)$ to $(-(M+1) + x_i, 0)$. Additionally, we add two segments aa' and bb' such that $a = (-1/2, 0)$, $a' = (-(M+1) - 1/2, 0)$, $b = (1/2, 0)$, and $b' = ((M+1) + 1/2, 0)$.

Any stabbing circle for S of minimum radius contains a, b in its interior. Thus the possibilities for such a stabbing circle are: If the associated disk contains $a, b, (x'_1, 0), \ldots, (x'_n, 0)$, or $(-(M+1)+x'_1, 0), \ldots, (-(M+1)+x'_n, 0), a, b$, then it has diameter $M+1/2$. If it contains $(-(M+1) + x'_{i+1}, 0), \ldots, (-(M+1) + x'_n, 0), a, b, (x'_1, 0), \ldots, (x'_i, 0)$ for $i < n$, then it has diameter $M + 1 - (x'_{i+1} - x_i)$. Since $\mathsf{MAXGAP}(X) \geq 1$, the stabbing circles of minimum radius belong to the last family. Thus $\mathsf{MAXGAP}(X)$ is equivalent to finding the stabbing circle for S of minimum radius.

The set S does not satisfy all the assumptions of this paper, since all endpoints are collinear and the segments are not pairwise disjoint. We construct a set S' obtained from S by translating every segment vertically by distinct values of at most $\varepsilon = 1/10$. Since ε is small compared to the difference between distinct values of diameters of different stabbing circles for S (which is at least $1/2$), a minimum stabbing circle for S' corresponds to a minimum stabbing circle for S which is combinatorially "the same". This proves that the lower bound also holds for the more restricted sets of segments considered in this paper. □

References

[1] M. Abellanas, F. Hurtado, C. Icking, R. Klein, E. Langetepe, L. Ma, B. Palop, and V. Sacristán. The farthest color Voronoi diagram and related problems. In *Proc. EuroCG'01*, pages 113–116.

[2] E. M. Arkin, J. M. Díaz-Báñez, F. Hurtado, P. Kumar, J. S. B. Mitchell, B. Palop, P. Pérez-Lantero, M. Saumell, and R. I. Silveira. Bichromatic 2-center of pairs of points. *Comput. Geom.*, 48(2):94–107, 2015.

[3] P. Cheilaris, E. Khramtcova, S. Langerman, and E. Papadopoulou. A randomized incremental approach for the Hausdorff Voronoi diagram of non-crossing clusters. *Algorithmica*, 2016. DOI 10.1007/s00453-016-0118-y.

[4] M. Claverol, E. Khramtcova, E. Papadopoulou, M. Saumell, and C. Seara. Stabbing circles for sets of segments in the plane. In *Proc. LATIN'16, LNCS 9644,* to appear.

[5] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, 4:311–336, 1989.

[6] H. Edelsbrunner, H. Maurer, F. Preparata, A. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, 22(3):274–281, 1982.

[7] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete Comput. Geom.*, 9(1):267–291, 1993.

[8] K. Mehlhorn, S. Meiser, and R. Rasch. Furthest site abstract Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 11(06):583–616, 2001.

[9] E. Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40(2):63–82, 2004.

[10] E. Papadopoulou and S. K. Dey. On the farthest line-segment Voronoi diagram. *Internat. J. Comput. Geom. Appl.*, 23(06):443–459, 2013.

Approximation of a Spherical Tessellation by the Laguerre Voronoi Diagram

Supanut Chaidee* Kokichi Sugihara*

Abstract

This paper presents a method for approximating spherical tessellations, the edges of which are geodesic arcs, using spherical Laguerre Voronoi diagrams. The approximation method involves fitting the polyhedron corresponding to the spherical Laguerre Voronoi diagram to the observed tessellation using optimization techniques.

1 Introduction

There are many natural phenomena that can be represented with polygonal patterns on a sphere, such as the patterns found on fruit skins. If the tessellation is similar to a Voronoi diagram, then a mathematical model can be constructed to assist with understanding the polygonal pattern formation.

Since the ordinary Voronoi diagram does not provide a good representation of most naturally occurring tessellations, consideration needs to be given to a weighting for the cells. One generalization of the Voronoi diagram is the Laguerre Voronoi diagram, a weighted Voronoi diagram, the edges of which are straight lines. This concept was introduced by [7, 2]. In brief, for a set S of n spheres $s_i = (\mathbf{x}_i, r_i)$ in $\mathbb{R}^d$, where $\mathbf{x}_i$ is the center of the sphere and r_i is the sphere radius, which is interpreted as the generator weight, the Laguerre distance of $\mathbf{x} \in \mathbb{R}^d$ from s_i is defined by

$$d_{\mathrm{L}}(\mathbf{x}, s_i) = \|\mathbf{x} - \mathbf{x}_i\|^2 - r_i^2.$$

This concept was extended to the spherical Laguerre Voronoi diagram (SLVD) in [8].

Active areas of research related to Voronoi diagrams are Voronoi recognition and approximation. The recognition problem is the determination of whether or not a tessellation is the Voronoi diagram. If it is not, we approximate it using the Voronoi diagram that provides the best fit. Many studies have also focused on planar tessellation. Recently, we proposed a method for fitting planar photographic images of spike-contining objects containing the generators in polygons, using ordinary spherical Voronoi diagrams, and applied it to fruit skin pattern analysis in [3].

*Graduate School of Advanced Mathematical Sciences, Meiji University, Japan, {schaidee, kokichis}@meiji.ac.jp

Regarding the Laguerre Voronoi diagram, Duan et al. studied a method for the recognition of planar tessellations in [6]. The SLVD recognition problem was recently proposed in [5] using the polyhedron corresponding to the SLVD. Some studies have focused on the 3D structure of the Laguerre approximation. The tessellation fitting problem, which was considered in [3], was solved using the SLVD in [4] by approximating the weights of the generators when the locations are known.

This study proposes a method for approximating the SLVD for a spherical tessellation when the generator locations cannot be derived by conventional methods, as in [5]. In this situation, it is necessary to approximate both generator locations and weights. The remainder of this paper is organized as follows. First, we recall some definitions and theorems related to the SLVD. The algorithms for recognizing the SLVD are then presented. For the case that the given tessellation is not represented exactly by the SLVD, the difference between the tessellation and the constructed SLVD is quantified, and an optimization method is employed to find the best fit SLVD. Since the SLVD corresponds to a convex polyhedron, the optimization is applied to adjust this polyhedron to fit the observed tessellation. Finally, we conduct experiments using simulated data to confirm the validity of our method.

2 Preliminaries

We assume that the tessellation and the SLVD are on the unit sphere U in $\mathbb{R}^3$, where the center of the sphere is located at the origin $O(0, 0, 0)$ of the Cartesian coordinate system.

We assume a tessellation $\mathcal{T} = \{T_1, ..., T_n\}$ consisting of n cells is a 3-regular spherical tessellation, where T_i is a convex spherical polygon, i.e., the polygon edges are sections of geodesic arcs.

Let p_i be a point on U. The sphere $\tilde{c}_i$ centered at p_i is defined by

$$\tilde{c}_i = \{p \in U | \tilde{d}(p_i, p) = r_i\},$$

where $\tilde{d}(p_i, p)$ is the geodesic distance between p_i and p. If $0 \leq r_i < \pi/2$, r_i is defined as the spherefs radius. Otherwise, r_i is the imaginary spherefs radius, whose details were given in [5].

The Laguerre Proximity, the distance used for Voronoi construction, is defined by

$$\tilde{d}_L(p, \tilde{c}_i) = \frac{\cos \tilde{d}(p, p_i)}{\cos r_i}.$$

The Laguerre bisector of two circles $\tilde{c}_i, \tilde{c}_j$ is defined by $B_L(\tilde{c}_i, \tilde{c}_j) = \{p \in U | \tilde{d}_L(p, \tilde{c}_i) = \tilde{d}_L(p, \tilde{c}_j)\}$.

For a set of n sphere circles $\tilde{G} = \{\tilde{c}_1, ..., \tilde{c}_n\}$ on U, the regions $L_i := \tilde{R}(\tilde{G}, \tilde{c}_i) = \{p \in U | \tilde{d}_L(p, \tilde{c}_i) < \tilde{d}_L(p, \tilde{c}_j), j \neq i\}$ for all i, including their boundaries, constitute the SLVD. We denote the SLVD with n regions $L_1, ..., L_n$ as $\mathcal{L} = \{L_1, ..., L_n\}$.

Sugihara presented algorithms for constructing the SLVD for a given set of sphere circles in [8]. Let $\pi(\tilde{c}_i)$ be the plane passing through $\tilde{c}_i$ and $H(\tilde{c}_i)$ the halfspace bounded by $\pi(\tilde{c}_i)$ including the origin of the sphere. The intersection of $\pi(\tilde{c}_i)$ and $\pi(\tilde{c}_j)$ is denoted by $\ell_{i,j}$. The following theorem gives the relation between $\ell_{i,j}$ and the Laguerre bisector.

Theorem 1 ([8]) *The bisector $B_L(\tilde{c}_i, \tilde{c}_j)$ is the intersection of U and the plane containing ℓ_{ij} and O.*

From Theorem 1, the SLVD can be constructed from the intersection of all halfspaces $H(\tilde{c}_i)$ for all i to obtain the convex polyhedron, and by projecting the edges of the polyhedron onto the sphere with respect to the center O.

3 SLVD Recognition

For a given SLVD $\mathcal{L}$, there exists a polyhedron $\mathcal{P}$ whose central projection onto the sphere coincides with $\mathcal{L}$. For any tessellation $\mathcal{T}$, we can construct the SLVD with the following algorithms, whose details were provided in [5].

Let $\mathcal{V}$ be the set of tessellation vertices. Let $\widehat{e}_{i,j}$ be the tessellation edge separating cells i and j, $P_{i,j}$ be the plane passing through the edge $\widehat{e}_{i,j}$, $v_{i,j,k}$ the tessellation vertex corresponding to cells i, j, k, and $P_i := \pi(\tilde{c}_i)$ of the i-th cell.

The following algorithm is for the construction of the first three planes in the recognition process.

Algorithm 1 [5]: Plane Construction with Three Adjacent Sites
Input: The sphere $\tilde{c}_i$ centered at $p_i(x_i, y_i, \sqrt{x_i^2 + y_i^2})$ with radius r_i, tessellation edges $\widehat{e}_{i,j}$, $\widehat{e}_{j,k}$, $\widehat{e}_{i,k}$, and tessellation vertex $v_{i,j,k}$.
Output: The three planes P_i, P_j, P_k with respect to polygons i, j, k.
Procedure:

1. Construct the plane P_i containing $\tilde{c}_i$.
2. Construct the planes $P_{i,j}$, $P_{i,k}$, $P_{j,k}$.
3. Find the intersections $\ell_{i,j}$ of P_i and $P_{i,j}$, and $\ell_{i,k}$ of P_i and $P_{i,k}$.
4. Construct a geodesic arc $\widehat{e}^c_{i,j}$ such that $\widehat{e}^c_{i,j}$ passes through p_i and is perpendicular to $\widehat{e}_{i,j}$.
5. Choose a point q_j in polygon j on the arc $\widehat{e}^c_{i,j}$.
6. Construct the plane P_j passing through $\ell_{i,j}$, q_j.
7. Find the intersection $\ell_{j,k}$ of the planes P_j and $P_{j,k}$.
8. Construct the plane P_k passing through the lines $\ell_{i,k}$ and $\ell_{j,k}$.

end Procedure

The following algorithm is for the generation of n planes for the tessellation.

Algorithm 2 [5]: Construction of n Planes
Input: Spherical tessellation $\mathcal{T}$ with tessellation vertices $\mathcal{V}$.
Output: The planes $P_1, ..., P_n$ with respect to the polygons $1, ..., n$.
Comment: $\mathbb{P}$ is the set of constructed planes.
Procedure:

1. make $\mathbb{P}$ empty;
2. Choose an arbitrary vertex $v_{i,j,k} \in \mathcal{V}$ and employ Algorithm 1 to construct planes P_i, P_j, P_k.
3. Add the planes P_i, P_j, P_k to the set $\mathbb{P}$.
4. Mark $v_{i,j,k}$ as a used vertex.
5. **while** there exists an unmarked vertex $v_{l,p,q} \in \mathcal{V}$ such that exactly two planes among P_l, P_p and P_q are included in $\mathbb{P}$.
   do
   Apply steps 2, 3,and 7 of Algorithm 1 to find $\ell_{l,q}$
   and $\ell_{p,q}$.
   Construct a plane P_q.
   Add P_q to the set $\mathbb{P}$.
   Mark the vertex $v_{l,p,q}$.
   end while

end Procedure

For each plane $P_i \in \mathbb{P}$, we consider the halfspace $H(P_i)$ which includes the sphere origin O, and find the intersection of all such halfspaces to obtain the convex polyhedron $\mathcal{P}$.

From Algorithm 1 the construction of the polyhedron $\mathcal{P}$ depends on the initial sphere and the point q_j. However, even if we choose them arbitrarily, the polyhedron construction can still be carried out. This can be formalized with the following theorem.

Theorem 2 ([5]) *For a given tessellation $\mathcal{T}$, the construction of a polyhedron corresponding to the SLVD $\mathcal{L}$ is possible with an arbitrary choice of the initial plane P_i in step 1 and the point q_j in step 5 of Algorithm 1.*

The proof of this theorem uses the transformation of a polyhedron in the projective space of $\mathbb{R}^3$ and the construction processes in Algorithm 1.

Let $\mathcal{L}$ be the tessellation obtained from the intersection of all halfspaces bounded by $P_1, ..P_n$ constructed by Algorithm 2 and projected onto a sphere. If the given tessellation $\mathcal{T}$ is exactly the SLVD, the tessellation $\mathcal{L}$ is identical to the tessellation $\mathcal{T}$, and arbitrary choice of Theorem 2 gives us the same SLVD. Otherwise, we get a different SLVD to $\mathcal{T}$.

The construction of the polyhedron $\mathcal{P}$ corresponding to the tessellation $\mathcal{T}$ shown in Algorithm 2 requires time complexity $O(n \log n)$.

4 SLVD Approximation Method

Note that almost all real world spherical tessellations cannot be represented exactly with an SLVD. In such instances, there exists a difference between $\mathcal{T}$ and $\mathcal{L}$. In this section, we define an index for this discrepancy, and provide a method for minimizing the discrepancy to obtain the best fit SLVD.

4.1 Discrepancy

For the tessellations $\mathcal{T}$ and $\mathcal{L}$, suppose that T_i corresponds to L_i for all i. For the i-th cell, let $A_i = T_i \cap L_i$ be the intersecting convex spherical polygon. Suppose that the polygon is k_i-gon, which we denote by the anticlockwise sequence of vertices $A_i = (A_{i,1}, ..., A_{i,k_i})$. Also, let $\alpha_{i,1}, ..., \alpha_{i,k_i}$ be the angles between two adjacent spherical k-gon edges. The area of the spherical polygon A_i is denoted by $\text{area}(A_i)$ If $A_i = \emptyset$, $\text{area}(A_i) = 0$. Otherwise, $\text{area}(A_i) = \sum_{j=1}^{k_i} \alpha_{i,j} - (k_i - 2)\pi$.

Let $A_\mathcal{T}, A_\mathcal{L}$ be the areas of spherical tessellations $\mathcal{T}$ and $\mathcal{L}$, respectively. The difference between the areas for the tessellations $\mathcal{T}$ and $\mathcal{L}$ are defined by $D_\mathcal{T} = A_\mathcal{T} - A$ and $D_\mathcal{L} = A_\mathcal{L} - A$, where $A = \sum_{i=1}^{n} \text{area}(A_i)$.

The discrepancy between $\mathcal{T}$ and $\mathcal{L}$ is defined by

$$\begin{aligned}\Delta_{\mathcal{T},\mathcal{L}} =& (D_\mathcal{T} + D_\mathcal{L})/(A_\mathcal{T} + A_\mathcal{L}) \\ =& 1 - \frac{1}{4\pi} \sum_{i=1}^{n} \left(\sum_{j=1}^{k_i} \alpha_{i,j} - (k_i - 2)\pi \right). \end{aligned} \quad (1)$$

4.2 The Procedure for Obtaining an SLVD Corresponding to a Given Tessellation

For the tessellation $\mathcal{T}$, we employ Algorithms 1 and 2 to construct the polyhedron and the SLVD. We compute the discrepancy by the following procedure.

1. For the tessellation $\mathcal{T} = \{T_1, ..., T_n\}$, determine the area of each cell. The set of all areas is denoted $\mathcal{A}_T = \{\text{area}(T_1), ..., \text{area}(T_n)\}$.
2. Choose the cell i such that $\text{area}(T_i) := \max \text{area}(T_j)$, $j = 1, ..., n$.
3. Starting from the i-th cell, define the center of the first generator from the centroid p_i of the cell. Define the weight of the cell as zero which, is the plane tangent to the i-th cell at p_i.
4. Without loss of generality, choose the location of the second generator from a point inside this cell.
5. Employ Algorithms 1 and 2 to construct a polyhedron $\mathcal{P}$ and project $\mathcal{P}$ onto the center of the sphere U.
6. For each i, find the intersection A_i of two spherical polygons T_i, L_i, and compute the discrepancy $D(\mathbf{x}) := \Delta_{\mathcal{T},\mathcal{L}}$.

4.3 Tessellation Fitting

To find the best fit SLVD, we find the minimum discrepancy from Equation (1). The discrepancy $\Delta_{\mathcal{T},\mathcal{L}}$ is related to the angle of the intersecting spherical polygons and the number of spherical polygon vertices which will change when the SLVD changes.

The main factor which affects the SLVD $\mathcal{L}$ is the alignment of planes $P_1, ..., P_n$ composing the polyhedron $\mathcal{P}$ of $\mathcal{L}$.

Let the plane P_i be

$$P_i : a_i x + b_i y + c_i z = d_i. \quad (2)$$

Since the plane P_i does not pass through the sphere's origin, the plane equation (2) can be expressed as

$$P_i : A_i x + B_i y + C_i z = 1. \quad (3)$$

The parameters A_i, B_i, C_i involve the alignment of the plane P_i. Therefore, the adjustment of the SLVD of n planes requires the parameters $\mathbf{x} = (A_1, ..., A_n, B_1, ..., B_n, C_1, ..., C_n)$.

We define the discrepancy function of $\mathbf{x}$ using the procedure in Section 4.2 by $D(\mathbf{x}) := \Delta_{\mathcal{T},\mathcal{L}}$.

However, it is complicated to find the relation between the planes and the angles as defined in (1). Therefore, we employ the Nelder-Mead method to find $\min D(\mathbf{x})$ numerically, where $D(\mathbf{x})$ is computed in a pointwise manner. The details of the method are provided in Chapter 18 of [1].

In brief, we first construct a simplex $\mathcal{S} = \{S_1, ..., S_{m+1}\}$ of m dimensional parameter space composed of $m+1$ vertices and compute the discrepancy function value for each simplex vertex. For each iteration, the worst vertex which yields the maximum discrepancy will be replaced with a new vertex by reflection, expansion, or contraction of the centroid points among the remaining m vertices with ratios $\alpha_R, \alpha_E, \alpha_C$. If we cannot replace the worst vertex, we shrink the simplex to the vertex that has the smallest discrepancy, with ratio α_S. Therefore, the direction of the simplex is moved to the local minimum of the discrepancy function. The iteration is terminated if it meets the convergence criteria.

For a tessellation of n cells, the number of parameters considered is $m = 3n$. Therefore, we consider a simplex of $3n+1$ vertices. The convergence condition is determined by the number of iterations.

We choose the first vertex S_1 of the initial simplex from the parameters of the planes constructed by the procedure provided in Section 4.2. The remaining $3n$ vertices are determined by $S_i = S_1 + \beta \mathbf{e}_i$, where $\mathbf{e}_i$ is the i-th standard basis of $\mathbb{R}^m$, and β is the positive number.

5 Experiments and Numerical Results

We conducted the experiments using simulated data. We used Wolfram Mathematica®10.3 to implement the algorithms.

To validate Algorithms 1 and 2, we generated the SLVD for the tessellation $\mathcal{T}$. From the experiments, we can find the tessellation $\mathcal{L}$ which coincides with the tessellation $\mathcal{T}$. The accuracy was measured using the discrepancy function value.

For the approximation, we separated the experiment into 2 parts, testing the validity of the framework, and fitting an SLVD to an arbitrary tessellation. The tessellation had 10 cells. We set $\alpha_R = 1, \alpha_E = 2, \alpha_C = -0.5, \alpha_S = -0.5$, and iterated 4,000 times.

We checked the validity of the approximation framework by generating the SLVD. After that, we perturbed some of the initial plane parameters, which yielded a different SLVD to the initial one. After that, we employed the Nelder-Mead method to optimize the discrepancy function. From the experiment, we found that we can find the local minimum that has the smallest discrepancy, and the estimated SLVD converges to the tessellation. The results for the discrepancy minimization are shown in Figure 1 (left).

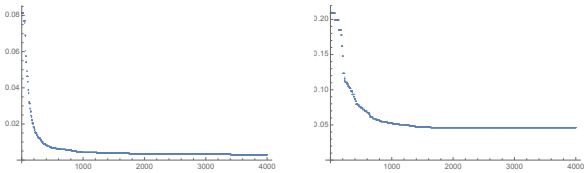


Figure 1: The change in the discrepancy of the initial simplex when (left) the initial simplex vertex was perturbed; (right) arbitrary spherical tessellation

After that, we conducted the experiment for an arbitrary spherical tessellation and employed the Nelder-Mead method. From the experiment, we can find the parameter that best fits the given tessellation. The change in the discrepancy is shown in Figure 1 (right), and the results for the fitted SLVD is shown in Figure 2.

6 Concluding Remarks

We proposed a framework for finding an SLVD that can be fitted to a given spherical tessellation. The optimization of the discrepancy function was shown to rely on the orientation of the polyhedron of the SLVD.

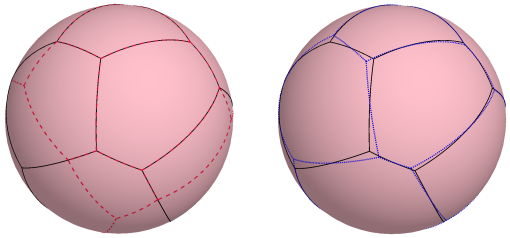


Figure 2: (Left) Solid lines: the spherical tessellation; dashed lines: SLVD from the S_1 parameter; (Right) dotted lines: the fitted SLVD from the optimization

The proposed framework can be used for recognizing whether the given tessellation is close to the Voronoi diagram. For the 3D real world spherical tessellation, we can extract the tessellation, project it onto a sphere and employ our framework. However, similar to [3, 4], this can be considered as a new problem when we use a planar photographic image instead of the information of 3D tessellation.

Acknowledgments

S.C. acknowledges the MIMS Ph.D. Program of the Meiji Institute for Advanced Study of Mathematical Sciences, Meiji University. This research was partly supported by the Grant-in-Aid for Basic Research No. 24360039 and Exploratory Research No.15K12067 of MEXT.

References

[1] Arora, J. S.: Introduction to optimum design (Third Edition). Elsevier, USA (2012)

[2] Aurenhammer F.: Power Diagram: Properties, Algorithms, and Applications, SIAM J. Comput., 16, 78–96 (1987)

[3] Chaidee S., and Sugihara K.: Approximation of Fruit Skin Patterns Using Spherical Voronoi Diagram, Pattern Anal Appl, DOI: 10.1007/s10044-016-0534-2 (2016)

[4] Chaidee S., and Sugihara K.: Fitting Spherical Laguerre Voronoi Diagrams to Real-World Tessellations Using Planar Photographic Images, accepted

[5] Chaidee S., and Sugihara K.: Recognition of the Spherical Laguerre Voronoi Diagram, in preparation

[6] Duan Q., Kroese D. P., Brereton T., Spettl A., Schmidt V.: Inverting Laguerre Tessellations, Comput. J. 57, 1431–1440 (2014)

[7] Imai H., Iri M., Murota K.: Voronoi Diagram in the Laguerre Geometry and its Applications. SIAM J. Comput. 14, 93–105 (1985)

[8] Sugihara K.: Laguerre Voronoi Diagram on the Sphere. J. Geom. Graph. 6, 69–81 (2002)

One Round Voronoi Game on Grids

Rebvar Hosseini * Mehdi Khosravian Mansoor Davoodi Bahram Sadeghi Bigham

Abstract

Recently there has been a great deal of interest in Voronoi Game: Two players insert a certain number of facilities in a determined number of rounds. The Voronoi Diagram of the inserted facilities is calculated and the winner is settled based on the Voronoi Region occupied by either of the players. A special version of the game in which the players insert their facilities in a single round is called "One Round Voronoi Game". Most of the previous studies in this area are performed in continuous game regions and facilities are considered as single points in the region with no area. In this paper, a new approach to One Round Voronoi Game is presented. Two players insert their facilities on a rectangular grid in one round. The area of the grid is shared between the players based on the *nearest neighbor rule* with *Manhattan* metric. Winning strategies are proposed for the first player in both one and two dimensional grids and the optimality of the strategy is proven in the one-dimensional case. Furthermore, the lower bound of winning margin is presented in both cases.

1 Introduction

Facility location is an optimization problem, concerning with placing a set of facilities which serve a set of customers based on an optimality measure. Adding *competitive market players* to this context and combining it with the arguments of *game theory* leads to the *competitive facility location* problem. This problem has been extensively studied in different fields such as computational geometry, mathematics, industrial engineering and operation research. The *Voronoi game* is a simple geometric model for the competitive facility location problem. From the viewpoint of rounds, there are two types of Voronoi game. In the *one round* version, the first player (White denoted by $\mathscr{W}$) places a set of k facilities in the game region, followed by the second player (Black denoted by $\mathscr{B}$) having the same number of facilities. In the other variation which is called *k-round* game, two players place one facility each alternately for k rounds in the game area.

*Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, rebvar@oulu.fi, {m_khosravian,mdmonfared,b_sadeghi_b}@iasbs.ac.ir

Voronoi game has been widely studied in the continuous space domain. One dimensional k-round Voronoi game where the game region is a line segment or a circle was studied by Ahn et al. [1]. The second player ($\mathscr{B}$) always wins the game by a winning margin of arbitrary small $\epsilon > 0$. Their defined k-round game is different from the *one* round game on the continues line segment where $\mathscr{W}$ can achieve a win by placing his facilities at the odd integer points. Also, similar to the k-round case, $\mathscr{W}$ can limit the loss margin as much as he wishes. Fekete and Meijer [2] proposed a model for two dimensional one round game played on a rectangular continuous demand region. They studied the winning conditions in terms of facility count and aspect ratio of the game board. The discrete Voronoi game was introduced by Teramoto et al. [3]. Two players place n facilities each in a graph which contains at least $2n$ nodes. They showed that in a complete $k-ary$ tree which is large enough with respect to n and k, the first player has a winning strategy. The Voronoi game on graphs and particularly on trees were later studied by Kiyomi et al. [4]. They showed that the game played on a path containing n vertexes and continued for $t < \frac{n}{2}$ rounds will end in a tie if either n is even or t is not one. When n is odd and $t = 1$, the first player wins the game. Banik et al. [5] studied another variation of the discrete Voronoi game which is played on a *simple polygon*. They proposed the complexity results when the number of facilities for each player is limited to one. They also studied one round discrete Voronoi game on a line segment [6]. In this problem, the players are competing for owning a set of n users by placing a set of m points each. They proved that if the sorted order of the n points on the line segment is known, the optimal strategy for the second player and first player can be computed in $O(n)$ and $O(n^{m-\lambda_m})$ respectively where $0 < \lambda_m < 1$ is a constant. Gerbner et al. [7] studied t-round voronoi game on graphs. They proved that there are graphs for which the second player gets almost all vertices in the game, but this is not possible for bounded-degree graphs. Further they showed that for trees, the first player can get at least one quarter of the vertices.

In this paper, we study the one round discrete Voronoi game on a grid $G(m, n)$. To achieve a better model, facilities are considered to have area. The problem is studied in one dimensional grid first and a winning strategy that guarantees the winning margin of one is

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

proposed for $\mathscr{W}$. Further, the optimality of $\mathscr{W}$'s strategy is shown as well. Two dimensional case in which the width of the grid, m, is an odd number is studied as well and condition of $\mathscr{W}$'s win are computed. These computations provide lower bounds in a way that $\mathscr{W}$ wins the game by a margin of m. It is clear that in the grid with even m, the symmetry play by $\mathscr{B}$ ends the game in a tie in most cases. However, proposing a winning strategy for even m seems much harder.
The rest of this paper is organized as follows: In the next section, the game definitions and formulation are presented. In Section 3, Voronoi game on the one dimensional grid is studied. The game in two dimensional grid board is discussed in Section 4. Finally, the last section summarizes some open issues which are introduced by this problem. Please see [8] for the complete proofs.

2 Voronoi Game on Grid

Grid Voronoi Game is denoted by $GVG_r(G,k)$ in which k is the number of facilities for either of the players and r is the number of play rounds. In the rest of this paper $G(m,n)$ is considered as the game play board. G is a rectangular *grid* with the length of n and the width of m and consists of $m \times n$ unit squares called *cell*s. All of the distances are measured using *Manhattan* metric. In the one round game variation ($r = 1$) each of the players (White denoted by $\mathscr{W}$ as the first player and Black denoted by $\mathscr{B}$ as the second player) chooses a set of k facilities disjoint from each other. One or both of the players will own the total area or a part of a cell respectively based on the nearest neighbor rule. Hence, the area of a cell which has the same distance from some cells occupied by $\mathscr{W}$ or $\mathscr{B}$, is shared among them. Furthermore, by placing a facility in a cell, the corresponding player will own all the area of that specific cell. The player owning the largest part of the region is the winner of the game.

3 One Dimensional Grid Voronoi Game

In this section, $G(1,n)$ is considered as a one dimensional grid with the length of n (and the width of $m = 1$). Without the loss of generality, suppose that the orientation of the grid is horizontal as illustrated in Figure 1.

Definition 1 *The distance between two consecutive inserted facilities of $\mathscr{W}$ is called an interval. The horizontal distance between the left side of the game region and the leftmost occupied cell by $\mathscr{W}$ is called left half interval and is denoted by LHI. Likewise, the half interval between the right side of the game region and the rightmost occupied cell by $\mathscr{W}$ is called right half interval and is denoted by RHI. The length of any full/half interval I is denoted by $|I|$.*

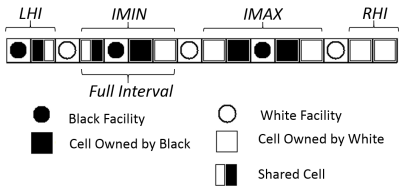


Figure 1: One dimensional grid Voronoi game $GVG_{r=1}(G(m=1,n=16),k=3)$

Considering the definitions we show that selecting the position of facilities according to

$$\left\lfloor \frac{(2i-1)\times n}{2k} \right\rfloor \quad ; i = 1, ..., k \tag{1}$$

in $GVG_1(G(1,n),k)$ is a winning strategy for $\mathscr{W}$. To prove it, the following propositions are required. Note that counting the grid cells is started from zero (see [8] for the extended versions and their proofs).

Proposition 1 *It is obvious that the distance between two optional cells is an integer number. If Eq. (1) is used the maximum length of a full interval in case of existence is $\lfloor \frac{n}{k} \rfloor$. An interval with the maximum length is denoted by IMAX. The minimum length of a full interval in case of existence is $\lfloor \frac{n}{k} \rfloor - 1$. IMIN indicates a full interval with the minimum length. For any n, $|RHI| \leq |LHI|$ holds. As a result $|RHI| + |LHI| \leq \lfloor \frac{n}{k} \rfloor$.*

Proposition 2 *$\mathscr{B}$ will own at least $|LHI|$ of the game region by placing a facility in an IMIN interval. This means that selecting LHI or RHI is dominated by the selection of an empty IMIN interval. Further, Placing two facilities in one IMIN or IMAX interval is not an efficient placing strategy for $\mathscr{B}$.*

Theorem 3 *$\mathscr{W}$ wins $GVG_1(G,k)$ in $G(1,n)$ by selecting the position of his facilities according to Eq. (1) where $2k \nmid n$. The game ends in a tie when $2k \mid n$.*

Proof. Assume that t is the number of IMIN intervals when $\mathscr{W}$ places his facilities according to Eq. (1). The number of IMAX intervals will be $k-1-t$. Considering Propositions 1 and 2, $\mathscr{B}$ is forced to place a facility in each interval and finally places a facility in LHI. Hence, the Voronoi region of $\mathscr{W}$ and $\mathscr{B}$ can be calculated. For the complete proof see [8]. □

3.1 Proof of Optimality

In this section, we prove that the placing based on Eq. (1) is an optimal placement strategy for $\mathscr{W}$. It is clear

that different arrangements of IMIN and IMAX intervals between LHI and RHI are also optimal placement strategies if placement based on Eq. (1) is optimal. The number of different ways to arrange t objects of one kind (IMAX intervals) and $k-1-t$ objects of another kind (IMIN intervals) in a row (all optimal placement strategies) is $(k-1)!/(t!(k-1-t)!)$. In the following Eq. (1) is used since different arrangements of IMIN and IMAX intervals are equivalent.

Theorem 4 *Placing facilities according to Eq.* (1) *is an optimal placement strategy for* $\mathscr{W}$.

Proof. Suppose that $\mathscr{W}$ uses an arbitrary placement strategy other than Eq. (1) (and its other equivalents). Also, denote the length of created half/full intervals by $L_0, L_1, ..., L_k$ from left to the right side of the grid. It is clear that by inserting a facility in each one of the white intervals, the difference between Voronoi region of $\mathscr{B}$ and $\mathscr{W}$ is

$$\mathrm{MIN}(L_0, L_k) - \mathrm{MAX}(L_0, L_k) + 1. \tag{2}$$

If $\mathrm{MIN}(L_0, L_k) \neq \mathrm{MAX}(L_0, L_k)$ is true, $\mathscr{B}$ will not lose the game (because $\mathrm{MIN}(L_0, L_k) \leq \mathrm{MAX}(L_0, L_k)$). As a result and since $|\mathrm{LHI}| = |\mathrm{RHI}|$ must holds, the loss margin of $\mathscr{B}$ is not more than one if he plays optimally. Also note in previous equations that the length of each interval is at least one. Otherwise, $\mathscr{B}$ always can achieve a tie by following the symmetry play (number of intervals is less than $k+1$). Now, suppose that the length of one of the intervals, I, is bigger than $|\mathrm{IMAX}|$ ($|I| = |\mathrm{IMAX}| + L$). We investigate this problem in two cases :$L \geq 2$ and $L = 1$. First, suppose that $L = 2$. $\mathscr{B}$ gains $\frac{|\mathrm{IMAX}|+3}{2}$ by placing one facility in this interval. Suppose that $L_0 = L_k < \lfloor \frac{n}{2k} \rfloor$. If $L_0 = L_k > \lfloor \frac{n}{2k} \rfloor$, placing a facility in an IMIN interval (there exist at least one if $k > 2$) is not efficient, because $|\mathrm{LHI}| = |\mathrm{RHI}| \geq \lfloor \frac{n}{2k} \rfloor > \frac{1}{2} \lfloor \frac{n}{k} \rfloor$ and as a result $|\mathrm{LHI}| = |\mathrm{RHI}| = \lfloor \frac{n}{2k} \rfloor$.
Since $\lfloor \frac{n}{2k} \rfloor < \frac{|\mathrm{IMAX}|+1}{2}$, placing two facilities in I when $|I| \geq |\mathrm{IMAX}| + 2$ guarantees equality for $\mathscr{B}$. The complete proof can be found in [8]. □

4 Two Dimensional Grid Voronoi Game

The game play scenario in two dimensional game is fundamentally different. Both of the players can freely choose the location of their facilities in two directions and as a result the winning strategies will change. Since the facilities in the grid Voronoi game have area, proposing winning strategy is much harder. Furthermore, in the grid Voronoi game more precise winning margin can be calculated and unlike the continuous case, none of the players can limit the loss margin arbitrarily. In the following, the winning condition for $\mathscr{W}$ will be discussed. Note however that, these conditions do not mean that $\mathscr{B}$ wins the game in the rest of cases (unlike the continuous region [2]). It is not difficult to show that $\mathscr{B}$ does not lose the game in the grid with even width (symmetry play in many cases). In this section suppose that $m \geq 3$ is an odd number. We denote the $\left(\frac{m+1}{2}\right)^{th}$ row of the grid by R_{mid} and we call it the *middle row*. Furthermore, similar to the one dimensional case, the horizontal distance between two consecutive facilities of $\mathscr{W}$ (which is a rectangle) is called an interval. In this section, assume that $\mathscr{W}$ will place his facilities according to Eq. (1) horizontally and in R_{mid} vertically. Therefore, the position of every facility of $\mathscr{W}$ is selected based on the following equation:

$$\left(\frac{m+1}{2}, \left\lfloor \frac{(2i-1) \times n}{2k} \right\rfloor\right) \; ; i = 1, ..., k. \tag{3}$$

Lemma 5 *Let* $n_1 = \frac{5}{3}m \times k - \frac{7}{3}k + 1$ *and* $\mathscr{W}$ *places his facilities in* $G(m, n)$ *according to Eq.* (3). *Also, suppose that* $\mathscr{B}$ *has placed a facility in* R_{mid} *in a full interval. For every* $n \geq n_1$, *this position is the most efficient place for the* $\mathscr{B}$*'s facility in that interval.*

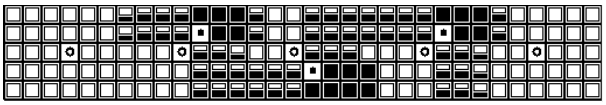
Lemma 6 *Assume that* $\mathscr{B}$ *places a facility in an interval* I *in a way that the total Voronoi region of that facility remains inside the bounds of* I. *Also suppose that the vertical distance from this facility to* R_{mid} *is* $a > 0$. *Transferring this facility vertically to* R_{mid} *will increase the Voronoi region and the amount of increment is* a^2.

Similar calculation for the case when the Voronoi region of a facility is in more than just one interval confirms the result of the previous lemmas. It is obvious now that for any $n \geq n_1$, moving a facility to another cell in the same interval decreases the Voronoi region for the facility (except for R_{mid}). But n_1 is not a tight lower bound (for example $GVG_1(G(7, 29), 3)$). Based on the number of cells which $\frac{1}{3}$ of them are owned by $\mathscr{B}$, it is easy to show that a lower bound for the width of the grid (for win margin of m) can be calculated as follows:

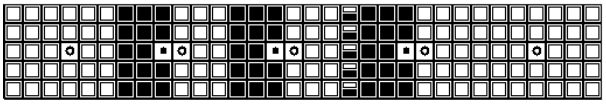
$$n_m = \begin{cases} n_1 & ; \left(\frac{m+1}{2}\right) \bmod 3 = 0 \\ n_1 - (k-2) & ; \left(\frac{m+1}{2}\right) \bmod 3 = 1 \\ n_1 & ; \left(\frac{m+1}{2}\right) \bmod 3 = 2 \end{cases} \tag{4}$$

This equation along with the previous lemmas, decreases the number of possible facility movements to two cases called *valid movements*.

- Transferring a facility from LHI to its neighboring interval (IMIN or IMAX) including the column containing $\mathscr{W}$'s facility.



(a) Zigzag strategy



(b) Simple strategy

Figure 2: Zigzag vs. Simple strategy.

- Transferring a facility from an IMIN interval to a neighboring IMAX one including the column containing $\mathscr{W}$'s facility.

Definition 2 *The intersection of the Voronoi regions of two facilities is called the overlapping of these facilities. Possible cases of overlapping (7 cases) are presented in [8].*

Lemma 7 *Suppose $G(m,n)$ is a grid in which $n \geq n_m$. $\mathscr{W}$ wins $GVG_1(G,2)$ with the winning margin of m if $2k \nmid n$. The game will end in a tie when $2k \mid n$.*

Lemma 8 *Let $G(m,n)$ be a grid in which $n \geq n_m$. $\mathscr{B}$ loses $GVG_1(G,3)$ with the minimum loss margin of m if $2k \nmid n$.*

We started to move the facilities by one of the valid movements. Similar calculations indicate that when a movement starts with a valid one it can only continue for at most three facility movements. Theorem 9 covers this problem.

Theorem 9 *For any odd m, any optional k and any $n \geq n_m$, $\mathscr{W}$ wins $GVG_1(G(m,n),k)$ with winning margin of m if $2k \nmid n$.*

Proof. It is clear that if $\mathscr{B}$ plays according to the simple strategy he loses the game by a loss margin of m. We are interested in the possibility of win or a smaller loss margin. To achieve either of these goals consider the first two facilities of $\mathscr{B}$. Assume that the Voronoi region of the first move by $\mathscr{B}$ is P' and the second one is Q'. Also suppose that by placing the same facilities in R_{mid} (according to the simple strategy), $\mathscr{B}$ gains P and Q respectively. It is clear that for a zigzag movement to be efficient, $|P'| + |Q'| > |P| + |Q|$. Considering this, for any k and m in a grid with $n = n_m$ a zigzag movement must start with one of the valid movements and only grows if these conditions hold. One first starts from the left-most facility of $\mathscr{B}$ and proceeds to the right side of the grid one interval at a time and checks whether one or both of the valid movements are possible. Assume that the first valid movement is possible starting from the left half interval. If $k = 2$ or $k = 3$ by Lemma 7 and Lemma 8 we know that $\mathscr{B}$ loses the game with a loss margin of m. Similar reasoning for $k > 3$ indicates that moving more than three consecutive facilities from R_{mid} starting with a first valid movement and independent of the neighboring intervals type is a non efficient action (Figure 2). Likewise, the second valid movement will become non efficient in at most three moves: the Zigzag movement of just two facilities is not efficient (for $k = 2$ in all cases). Similarly, three movements in all cases are non efficient. □

5 Conclusion and Future Works

An optimal winning strategy for White (the first player) in both one and two dimensional grids is proposed. Like other variations of the Voronoi game problem several questions arise in this context as well. The most interesting problem is probably the case of a grid with even width. Showing that $\mathscr{B}$ does not lose is not difficult. $\mathscr{B}$ can gain at least half of the game region in most cases by *symmetry play* (not possible in all cases). Two dimensional k-round game which is a challenging problem in most contexts is an interesting open problem as well.

References

[1] Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M., van Oostrum. *Competitive facility location: the Voronoi game.* Theoretical Computer Science 310.1 (2004): 457-467.

[2] Fekete, Sándor P., and Henk Meijer. *The one-round Voronoi game replayed.* Computational Geometry 30.2 (2005): 81-94.

[3] Teramoto, Sachio, Erik D. Demaine, and Ryuhei Uehara. *Voronoi game on graphs and its complexity.* Computational Intelligence and Games, 2006 IEEE Symposium on. IEEE, 2006.

[4] Kiyomi, Masashi, Toshiki Saitoh, and Ryuhei Uehara. *Voronoi game on a path.* IEICE Transactions on Information and Systems 94.6 (2011): 1185-1189.

[5] Banik, Aritra, Sandip Das, Anil Maheshwari, and Michiel Smid. *The discrete voronoi game in a simple polygon.* Computing and Combinatorics. Springer Berlin Heidelberg, 2013, 197-207.

[6] Banik, Aritra, Bhaswar B. Bhattacharya, and Sandip Das. *Optimal strategies for the one-round discrete Voronoi game on a line.* Journal of Combinatorial Optimization 26.4 (2013): 655-669.

[7] Gerbner, Dániel and Mészáros, Viola and Pálvölgyi, Dömötör and Pokrovskiy, Alexey and Rote, Günter. *Advantage in the discrete Voronoi game.* J. Graph Algorithms Appl. 18.3 (2014): 439-457.

[8] Technical Report *One-Round Voronoi Game on Grids* - `http://rebvar.nabzekala.com/files/voronoi.pdf`

Generalized Colorful Linear Programming and Further Applications

Frédéric Meunier* Wolfgang Mulzer† Pauline Sarrabezolles* Yannik Stein†

Abstract

Colorful linear programming (CLP) is a generalization of linear programming that was introduced by Bárány and Onn. Given k point sets $C_1, \dots, C_k \subset \mathbb{R}^d$ that each contain a point $\boldsymbol{b} \in \mathbb{R}^d$ in their positive span, the problem is to compute a set $C \subseteq C_1 \cup \dots \cup C_k$ that contains at most one point from each set C_i and that also contains $\boldsymbol{b}$ in its positive span, or to state that no such set exists. CLP is known to be NP-hard.

We consider a generalization of CLP in which we are given additionally for each set C_i a number $l_i \in \mathbb{N}$, $i = 1, \dots, k$, and we want to find a set that contains at most l_i points from C_i. We call this problem *generalized colorful linear programming* (GCLP). While we show that even seemingly simple cases of GCLP remain NP-hard, we present a weakly-polynomial algorithm for the special case that there are only two colors and that the vectors of each set C_i contain $\boldsymbol{b}$ in their positive span. This case is particularly interesting due to its connection with the colorful Carathéodory theorem. Furthermore, we consider additional applications of CLP to problems on colored graphs.

1 Introduction

The colorful Carathéodory theorem [2] states that given $C_1, \dots, C_{d+1} \subset \mathbb{R}^d$ point sets that all contain the origin in their convex hulls, there always exists a set $C \subset C_1 \cup \dots \cup C_{d+1}$ that contains at most one point from each set C_i, $i = 1, \dots, d+1$, and that also contains the origin in its convex hull. We call the sets C_i, $i = 1, \dots, d+1$, *color classes* and we call a set with at most one point from each color class a *colorful choice*. Bárány also gave the following more general version.

Theorem 1 ([2]) *Let $C_1, \dots, C_d \subset \mathbb{R}^d$ be point sets and $\boldsymbol{b} \in \mathbb{R}^d$ a point with $\boldsymbol{b} \in \operatorname{pos}(C_i)$, for $i = 1, \dots, d$. Then, there is a colorful choice C with $\boldsymbol{b} \in \operatorname{pos}(C)$.*

Here, we denote with $\operatorname{pos}(P) = \{\sum_{\boldsymbol{p}_i \in P} \alpha_i \boldsymbol{p}_i \mid \alpha_i \geq 0 \text{ for all } \boldsymbol{p}_i \in P\}$ for a set $P \subset \mathbb{R}^d$ the set of all nonnegative linear combinations of points in P. Using a simple lifting argument, it can be shown that Theorem 1 implies the classic (convex) version of the colorful Carathéodory theorem as stated in the beginning.

In the spirit of the colorful Carathéodory theorem, Bárány and Onn [3] generalized linear programming to the colorful setting: given a point $\boldsymbol{b} \in \mathbb{R}^d$ and point sets $C_1, \dots, C_k \subset \mathbb{R}^d$, we want to find a colorful choice C with $\boldsymbol{b} \in \operatorname{pos}(C)$ or state that there is none. We call this problem *colorful linear programming* (CLP) and we call the decision problem to decide whether there exists such a colorful choice DCLP. Bárány and Onn [3] showed that DCLP is NP-complete even if $k = d$ and each C_i contains $\mathbf{0}$ in its convex hull. This was extended by Mulzer and Stein [8] who showed that DCLP is NP-complete even if $k = d+1$ and each C_i does not necessarily contain $\mathbf{0}$ in its convex hull, and by Meunier and Sarrabezolles [7] who showed that DCLP is NP-complete for all values of k if each C_i does not necessarily contain $\mathbf{0}$ in its convex hull. We define the following generalization of CLP (GCLP): given a point $\boldsymbol{b} \in \mathbb{R}^d$, point sets $C_1, \dots, C_k \subset \mathbb{R}^d$, and numbers $l_1, \dots, l_k \in \mathbb{N}$, we want to find a set C such that $|C \cap C_i| \leq l_i$ for $i \in [k]$ and such that $\boldsymbol{b} \in \operatorname{pos}(C)$ or state that there is none. We obtain CLP by setting $l_1 = \dots = l_k = 1$.

Since CLP is NP-hard, GCLP is NP-hard as well. However, as with regular linear programming and integer programming, GCLP is very versatile and can be used to model colorful versions of many combinatorial problems. Therefore, it is of interest to identify special cases of GCLP that can be solved in polynomial time or to show that even the more restricted version of the problem remains NP-hard. We consider several such examples and delineate a more precise boundary between easy and hard colorful problems.

2 Generalized Colorful Linear Programming

In CLP, we want to find a set that contains at most one point from each color class. In *generalized colorful linear programming* (GCLP) we allow additionally to be given k nonnegative integers $l_1, \dots, l_k$ that determine the number of points that we are allowed to take from each color class. We call a set C with $|C \cap C_i| \leq l_i$ for $i \in [k]$ an $(l_1, \dots, l_k)$-*colorful choice* or (with a slight abuse of notation) just a *colorful choice*.

*Université Paris Est, CERMICS (ENPC), {frederic.meunier,pauline.sarrabezolles}@enpc.fr.

†Institut für Informatik, Freie Universität Berlin, {mulzer,yannikstein}@inf.fu-berlin.de. WS was supported in part by DFG Grants MU 3501/1 and MU 3501/2. YS was supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

2.1 Complexity

Since GCLP is a generalization of CLP, it remains NP-hard. However, even seemingly simple special cases such as $k = 1 \wedge l_1 = d$ [3, 6] or $k = 2 \wedge l_1 = l_2 = d/2$ [3] have been show to be NP-hard as well. We show that even if the number of colors is fixed and each l_i is a constant fraction of $|C_i|$, $i \in [k]$, the problem remains NP-hard. We prove this for the convex version of GCLP. That is, we want to find a colorful choice C that contains $\boldsymbol{b}$ in its convex hull instead of just in its positive span. Without loss of generality, we can assume $\boldsymbol{b} = \mathbf{0}$. By a lifting argument, it can be easily shown that the convex version of GCLP is a special case of GCLP as stated in the introduction. Hence, any hardness results for the convex version hold for the cone version as well. The following theorem is the main tool in the reduction. The theorem was first obtained by Knauer et al. [6], albeit with a different proof. We compare both proofs below.

Theorem 2 *Let $P \subset \mathbb{R}^d$ be a set of size $2d$. It is NP-complete to decide whether there is a subset $P' \subset P$ of size d containing the origin in its convex hull.*

Proof. Let $A = \{a_1, \ldots, a_d\}$ be an instance of Partition, for d even. For $i \in \{1, \ldots, d-1\}$, we define the vector $\boldsymbol{v}_i \in \mathbb{R}^d$ as having its ith coordinate equal to 1, its last coordinate equal to a_i, and all other coordinates equal to 0. The vector $\boldsymbol{v}_d$ has all its coordinates equal to -1 except for the last coordinate, which is equal to a_d. Similarly, we define vectors $\boldsymbol{w}_i \in \mathbb{R}^d$, and just replace the last coordinate by $-a_i$. Assume there is a partition A_1, A_2 of A with $\sum_{a \in A_1} a = \sum_{a \in A_2} a$. Then, we have $\sum_{a_i \in A_1} \boldsymbol{v}_i = -\sum_{a_i \in A_2} \boldsymbol{w}_i$ and hence $\mathbf{0} \in \operatorname{conv}(\{\boldsymbol{v}_i \mid a_i \in A_1\} \cup \{\boldsymbol{w}_i \mid a_i \in A_2\})$. On the other hand, let $V' \subseteq \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d\}$ and $W' \subseteq \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_d\}$ be s.t. $|V'| + |W'| = d$ and s.t. $\mathbf{0} = \sum_{\boldsymbol{v} \in V'} \lambda_{\boldsymbol{v}} \boldsymbol{v} + \sum_{\boldsymbol{w} \in W'} \lambda_{\boldsymbol{w}} \boldsymbol{w}$, where $\sum_{\boldsymbol{v} \in V'} \lambda_{\boldsymbol{v}} + \sum_{\boldsymbol{w} \in W'} \lambda_{\boldsymbol{w}} = 1$ and $\lambda_{\boldsymbol{v}}, \lambda_{\boldsymbol{w}} \geq 0$ for all $\boldsymbol{v} \in V', \boldsymbol{w} \in W'$. By construction, for all $i = 1, \ldots, d$, we have either $\boldsymbol{v}_i \in V'$ or $\boldsymbol{w}_i \in W'$ and furthermore, all coefficients $\lambda_{\boldsymbol{v}}, \lambda_{\boldsymbol{w}}$, $\boldsymbol{v} \in V'$, $\boldsymbol{w} \in W'$, are equal. Hence, the sets $A_1 = \{a_i \mid \boldsymbol{v}_i \in V'\}$, $A_2 = \{a_i \mid \boldsymbol{w}_i \in W'\}$ form a partition of A with $\sum_{a \in A_1} a = \sum_{a \in A_2} a$. □

We note that the set P constructed in the proof of Theorem 2 was first described by Bárány and Onn [3, Theorem 5.1]. However, they used it to prove the weaker statement *DCLP is NP-complete even for $k = d$*. This result is a consequence from Theorem 2 by setting $C_1 = \cdots = C_d = P$. Also, NP-hardness of the two special cases $k = 1 \wedge l_1 = d$ and $k = 2 \wedge l_1 = l_2 = d/2$ follows from Theorem 2 by setting $C_1 = P \wedge l_1 = d$ and $C_1 = C_2 = P \wedge l_1 = l_2 = d/2$, respectively.

Note further that the problem from Theorem 2 was first shown to be NP-complete by Knauer et al. [6]. Additionally, the proof of Theorem 2 gives an alternative proof for the #P-completeness of computing the simplicial depth. This hardness result was first obtained by Afshani et al. [1] and the alternative reduction is analogous to the proof of [1, Theorem 9]. It is not immediate that the reduction from Knauer et al. [6] has similar implications.

In the following, let $\text{GCLP}_k(r_1, \ldots, r_k)$, $r_i \in (0, 1)$, denote GCLP restricted to instances with exactly k color classes and the l_i's are given by $l_i = \lceil r_i n_i \rceil$ for $i \in [k]$, where $n_i = |C_i|$. That is, we are allowed to take a constant fraction of each color class.

Theorem 3 *For any fixed $k \in \mathbb{N}$ and any fixed ratios $r_1, \ldots, r_k \in (0, 1)$, $GCLP_k(r_1, \ldots, r_k)$ is NP-hard.*

Proof. We prove the statement by a reduction similar to the proof of Theorem 2. Given some partition instance $A = \{a_1, \ldots, a_d\}$, let $P \subset \mathbb{R}^d$, $|P| = 2d$, denote the same point set as in the proof of Theorem 2. If $r_1|P| = d$, we set $C_1 = P$ and create "dummy" points for $C_2, \ldots, C_k$ that will never be part of a convex combination of $\mathbf{0}$. To ensure this, we lift P to $\mathbb{R}^{d+1}$ by appending a 0-coordinate. Now, we set $C_i = \{\boldsymbol{c}_i\}$ for $i = 2, \ldots, k$, where the coordinates of $\boldsymbol{c}_i \in \mathbb{R}^{d+1}$ are 0 in dimensions $j = 1, \ldots, d$ and some positive number in dimension $d + 1$.

Now, assume $\lceil r_1|P| \rceil < d$ and hence $|P| < d/r_1$. We add $\lfloor d/r_1 - |P| \rfloor$ dummy points together with P to C_1 and create the other color classes as before. Then, we have $\lceil r_1|C_1| \rceil = d$ as desired.

The last case is $\lceil r_1|P| \rceil > d$. Again, we set $C_1 = P$ and construct $C_2, \ldots, C_k$ as above. To ensure that we only take d points from P, we add "mandatory" points to C_1 that have to be part of any convex combination of $\mathbf{0}$. We construct a mandatory point $\boldsymbol{q}$ by introducing a new dimension in which we set the coordinates of all other points to 1. The new point $\boldsymbol{q}$ has coordinates set to 0 in all but the new dimension, where it is set to -1. A short calculation reveals that we have to add $m = \left\lfloor \frac{r_1|P| - d}{1 - r_1} \right\rfloor$ mandatory points together with P to C_1 in order to ensure that $\lceil r_1|C_1| \rceil = d + m$.

Thus, the existence of a $(r_1|C_1|, \ldots, r_k|C_k|)$-colorful choice is equivalent to the existence of a partition of A into two sets A_1, A_2 with $\sum_{a \in A_1} a = \sum_{a \in A_2} a$. Since r_1 is constant, we can create the additional dummy/mandatory points in polynomial time. □

2.2 A Special Case

We now consider the following special case of GCLP: given a point $\boldsymbol{b} \in \mathbb{R}^d$, a ratio $r \in [0, 1]$, and point sets $C_1, C_2 \subset \mathbb{R}^d$ of size d with $\boldsymbol{b} \in \operatorname{pos}(C_i)$ for $i = 1, 2$, we want to find an $(\lceil rd \rceil, \lfloor rd \rfloor)$-colorful choice C with $\boldsymbol{b} \in \operatorname{pos}(C)$, or state that there is none.

Theorem 1 guarantees the existence of such a colorful choice: we set the first $\lceil rd \rceil$ color classes to copies

of C_1, and the next $\lfloor rd \rfloor$ color classes to copies of C_2. Hence, this simple case of only two colors is particularly interesting as we know that there always exists a solution, but computing it is already nontrivial. Note that for $l_1 = \lceil rd \rceil - 1$ or $l_2 = \lfloor rd \rfloor - 1$ the problem becomes NP-hard as a consequence of Theorem 2.

We give a weakly-polynomial algorithm for the two-color case that is based on constructing a family of linear programs. Let L denote the linear system $A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$, where $A \in \mathbb{R}^{d \times 2d}$ contains C_1 as its first d columns and C_2 as its second d columns. In the following, we assume that L is in general position. Given a cost vector $\boldsymbol{c} \in \mathbb{R}^d$, we denote with $L_{\boldsymbol{c}}$ the linear program that maximizes the objective function $\boldsymbol{c}^T\boldsymbol{x}$ subject to the equalities and inequalities from L. Let $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ be two generic cost vectors such that C_1 and C_2 are optimal bases. One can show that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ can be obtained in polynomial time. For $\lambda \in [0,1]$, we denote with $\boldsymbol{c}_\lambda$ the cost vector $\lambda\boldsymbol{c}_1 + (1-\lambda)\boldsymbol{c}_2$ and with L_λ the linear program $L_{\boldsymbol{c}_\lambda}$. That is, the linear programs L_λ, $\lambda \in [0,1]$, differ only in their cost functions which are convex combinations of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Our construction has the following properties.

Lemma 4 *There is a finite number of ordered intervals $I_1, \ldots, I_s$ with pairwise disjoint interiors such that $\bigcup_{i=1}^{s} I_i = [0,1]$ and such that*

(i) The length of each interval I_i, $i \in [s]$, is at least $1/K$, where $K \in \mathbb{N}$ and $\log K$ is bounded by a polynomial in the description size of L.

(ii) For each $i \in \{1, \ldots, s\}$, there is a unique feasible basis that is optimal for all L_λ, where λ is contained in the interior of I_i.

(iii) For λ belonging to two distinct intervals I_i, I_{i+1}, there are exactly two optimal bases that differ exactly by one column.

Proof. (i): This follows from standard tools such as Cramer's rule and the Leibniz formula for determinants. (ii) & (iii): Let $\lambda \in [0,1]$ and let B be an optimal basis for L_λ. We denote with N the set of columns from A not in B. Then, the *reduced cost vector* [5] is given by $\boldsymbol{r}_{B,\lambda} = (\boldsymbol{c}_\lambda)_N - A_N^T(A_B^{-1})^T$, where $(\boldsymbol{c}_\lambda)_N$ denotes the subvector of $\boldsymbol{c}_\lambda$ restricted to the coordinates corresponding to columns in N, A_N denotes the submatrix of A with columns in N, and A_B denotes the submatrix of A with columns in B. If the sign of the ith coordinate of $\boldsymbol{r}_{B,\lambda}$ is positive, then swapping the corresponding column from N into B increases the cost and otherwise (if the sign is non-positive), the cost remains equal or decreases. Since we want to maximize the objective function, a basis is optimal iff all coordinates of $\boldsymbol{r}_{B,\lambda}$ are non-positive, and it is unique if all coordinates of $\boldsymbol{r}_{B,\lambda}$ are negative.

We obtain the intervals $I_1, \ldots, I_s$ iteratively as follows: initially we set $\lambda = 0$. By general position and genericity of $\boldsymbol{c}_1$, the unique optimal basis for L_λ is C_1, i.e., all coordinates of $\boldsymbol{r}_{B,\lambda}$ are negative. Now, we continuously increase λ until one of the coordinates of $\boldsymbol{r}_{B,\lambda}$ becomes 0. Let λ_1 denote this value and let i be the coordinate of $\boldsymbol{r}_{B,\lambda_1}$ that is 0 (by general position and genericity, i is unique). Since C_1 is not an optimal basis for L_1, λ_1 exists. Because each coordinate of $\boldsymbol{r}_{b,\lambda}$ is a linear function in λ, $(\boldsymbol{r}_{b,\lambda'})_i$ is positive for all $\lambda' > \lambda_1$. Then, there exists an $\varepsilon > 0$ such that i is the only nonnegative coordinate of $\boldsymbol{r}_{b,\lambda'}$ for $\lambda' \in I = (\lambda_1, \lambda_1 + \varepsilon)$. Hence, for all $\lambda' \in I$, the basis B' that is obtained by swapping the column from N that corresponds to coordinate i of $\boldsymbol{r}_{b,\lambda}$ into B is the unique optimal basis. Note further, that both B' and B are optimal for L_{λ_1}. Set $I_1 = [0, \lambda_1]$ and construct iteratively the next intervals until $B' = C_2$. Let $\lambda_s \in [0,1]$ be the minimum value for which C_2 is an optimal basis for L_{λ_s}. Then, C_2 is optimal for every $\lambda' \in [\lambda_s, 1]$. We set $I_s = [\lambda_s, 1]$ and conclude the construction of the intervals. □

We now describe the complete algorithm. In round i, we maintain an interval $[a_i, b_i] \subset [0,1]$, such that the optimal basis for L_{a_i} contains at least $\lceil rd \rceil$ columns from C_1 (and due to the general position assumption, at most $\lfloor (1-r)d \rfloor$ columns from C_2) and such that the optimal basis for L_{b_i} contains at most $\lceil rd \rceil$ columns from C_1. We maintain the following invariant: there exists a $\lambda \in [a_i, b_i]$ such that the optimal basis for L_λ is the desired $(\lceil rd \rceil, \lfloor rd \rfloor)$-colorful choice.

Initially, we set $[a_1, b_1] = [0,1]$. By definition, C_1 is the optimal basis for L_0 and C_2 is the optimal basis for L_1. By Lemma 4(iii) optimal bases for two adjacent intervals differ only in one column, and hence the invariant holds for $[a_1, b_1]$. We solve then the linear program L_λ for $\lambda = (a_k + b_k)/2$ and let B^* denote the optimal basis. If B^* contains at least $\lceil rd \rceil$ columns from C_1, we set a_{i+1} to λ and $b_{i+1} = b_i$. Otherwise, we set $a_{k+1} = a_k$ and $b_{k+1} = \lambda$. Let B_1 be the optimal basis for $L_{a_{i+1}}$ and let B_2 be the optimal basis for $L_{b_{i+1}}$. Since B_1 contains at least $\lceil rd \rceil$ columns of C_1 and since B_2 contains at most $\lceil rd \rceil$ columns of C_1, the invariant holds for $[a_{i+1}, b_{i+1}]$ again by Lemma 4 (iii).

After $i^* = O(\log K)$ iterations, the interval $[a_{i^*}, b_{i^*}]$ is contained in the union of two adjacent intervals I_j, I_{j+1} with $j \in [s-1]$. Let B_j and B_{j+1} be the optimal bases for I_j and I_{j+1}, respectively. Hence, by Lemma 4 (ii), either B_j or B_{j+1} is the desired basis.

Each round requires polynomial time, and the number of rounds is bounded by a polynomial in the bit-size of the input. The following theorem is immediate.

Theorem 5 *Let $\boldsymbol{b} \in \mathbb{R}^d$ be a vector and let $C_1, C_2 \subset \mathbb{R}^d$ be two sets of size d with $\boldsymbol{b} \in \text{pos}(C_i)$ for $i = 1, 2$. Furthermore, let $r \in [0,1]$ be a parameter. Then, there is an algorithm that computes an $(\lceil rd \rceil, \lfloor rd \rfloor)$-colorful choice C with $\boldsymbol{b} \in \text{pos}(C)$ in weakly-polynomial time.*

3 Applications of Colorful Linear Programming

We consider two problems on colored graphs that can be cast as a CLP and analyze their complexity. The first problem is called ColorfulPath: given a directed graph $G = (V, E)$ whose edges are partitioned into k color classes $C_1, \ldots, C_k$ and two vertices $s, t \in V$, the problem is to decide whether there exists a directed path from s to t with at most one edge from each color class. ColorfulPath is a special case of CLP, since the existence of an s-t path can be modeled as a flow. Chakraborty et al. [4] showed this problem to be NP-complete by a reduction from 3-SAT. We present a similar but simplified proof, that reduces the number of necessary colors from $O(mn^2)$ to $O(m)$, where m is the number of clauses and n is the number of variables in the 3-SAT formula.

Theorem 6 ColorfulPath *is* NP*-complete, even if the graph $G = (V, E)$ is acyclic and $|E| = O(|V|)$.*

Proof. Consider a 3-SAT formula Φ, with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$, each containing exactly three literals. Our directed graph has $3m$ colors c_{jk}, $j = 1, \ldots, m$ and $k = 1, 2, 3$, one for each literal in each clause. We allow multiple edges between two vertices. However, our construction can be easily modified to at most one edge per vertex-pair by introducing new vertices. For each clause C_j we have one clause gadget G_j and for each variable x_i, we have one variable gadget G'_i. The clause gadget G_j for a clause C_j consists of two vertices $\{s_j, t_j\}$ and three directed edges from s_j to t_j with colors c_{j1}, c_{j2}, and c_{j3}. The variable gadget G'_i for a variable x_i consists of two edge-disjoint paths that are vertex disjoint except at the start and the end vertex. The first path contains one edge for each positive occurrence of x_i in Φ, colored with the color that corresponds to this literal. The second path contains one analogous edge for each negative occurrence of x_i in Φ. The graph G is obtained by concatenating all clause gadgets and all vertex gadgets and by identifying the last vertex in each gadget with the first vertex in the following gadget. This construction can be performed in polynomial time, and there is a colorful path through all gadgets if and only if Φ is satisfiable. □

We conclude with AnotherColorfulCycle (ACC): given a graph $G = (V, E)$, where $|E| = 2|V|$ and all edges are colored with $n = |V|$ colors such that exactly two edges have the same color, and a colorful Hamilton cycle in G, we want to find another colorful cycle (not necessarily Hamiltonian). This is a special case of the PPAD-complete problem AnotherColorfulSimplex [7] (ACS) and related to the PPA-problem AnotherHamiltonPath [9] (AHP), in which we are given a graph G and a Hamilton path in G, and we want to find another Hamilton path in G or in its complement. While there are no polynomial-time algorithms known for ACS and AHP, we show that ACC can be solved efficiently.

Theorem 7 AnotherColorfulCycle *can be solved in polynomial time.*

Proof. Consider the bipartite graph $G' = (V', E')$ with vertices $V' = V \cup \{C_1, \ldots, C_n\}$. There is an edge $(v, C_i) \in E'$ if there is an outgoing edge from a vertex $v \in V$ with color C_i in G. Note that there is a bijection between E' and E. Furthermore, the edges $M \subset E'$ in G' that correspond to the edges of the Hamiltonian cycle in G are a perfect matching in G'.

Since $|E| \geq |V|$, there is a cycle C in G'. As each vertex $C_i \in V'$, $i \in [n]$, is incident to two edges, and since one of them is contained in M, C is of even length and its edges alternate between M and $E' \setminus M$. Then, $M' = M \oplus C$ is a perfect matching different from M. It induces a colorful set of edges where each vertex $v \in V$ has exactly one outgoing edge in M'. Hence, M' corresponds to a colorful cycle in G. □

References

[1] Peyman Afshani, Donald R. Sheehy, and Yannik Stein. Approximating the simplicial depth in high dimensions. In *EWCG 2016*.

[2] Imre Bárány. A generalization of Carathéodory's theorem. *Discrete Mathematics*, 40:141–152, 1982.

[3] Imre Bárány and Shmuel Onn. Colourful linear programming and its relatives. *Mathematics of Operations Research*, 22:550–567, 1997.

[4] Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Raphael Yuster. Hardness and algorithms for rainbow connection. *Journal of Combinatorial Optimization*, 21(3):330–347, 2011.

[5] Vašek Chvátal. *Linear programming*. Macmillan, 1983.

[6] Christian Knauer, Hans Raj Tiwary, and Daniel Werner. On the computational complexity of ham-sandwich cuts, Helly sets, and related problems. In *STACS 2011*.

[7] Frédéric Meunier and Pauline Sarrabezolles. Colorful linear programming, Nash equilibrium, and pivots. *CoRR*, abs/1409.3436, 2014.

[8] Wolfgang Mulzer and Yannik Stein. Computational Aspects of the Colorful Carathéodory Theorem. In *SoCG 2015*.

[9] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48:498–532, 1994.

Random Sampling with Removal

Bernd Gärtner * Johannes Lengler * May Szedlák *

Abstract

Random sampling is a classical tool in constrained optimization. Under favorable conditions, the optimal solution subject to a small subset of randomly chosen constraints violates only a small subset of the remaining constraints. Here we study the following variant that we call random sampling with removal: suppose that after sampling the subset, we remove a fixed number of constraints from the sample, according to an arbitrary rule. Is it still true that the optimal solution of the reduced sample violates only a small subset of the constraints? The question naturally comes up in situations where the solution subject to the sampled constraints is used as an approximate solution to the original problem.

We study random sampling with removal in a generalized, completely abstract setting where we assign to each subset R of the constraints an arbitrary set $V(R)$ of constraints disjoint from R; in applications, $V(R)$ corresponds to the constraints violated by the optimal solution subject to only the constraints in R. Furthermore, our results are parametrized by the dimension δ, i.e., we assume that every set R has a subset B of size at most δ with the same set of violated constraints. This is the first time this generalized setting is studied.

In this setting, we prove matching upper and lower bounds for the expected number of constraints violated by a random sample, after the removal of k elements. For a large range of values of k, the new upper bounds improve the previously best bounds for LP-type problems, which moreover had only been known in special cases. We show that this bound on special LP-type problems can be derived in the much more general setting of violator spaces, and with very elementary proofs.

1 Introduction

On a high level, random sampling can be described as an efficient way of learning something about a problem, by first solving a subproblem of much smaller size. A classical example is the problem of finding the smallest element in a *sorted compact list* [2, Problem 11-3]. Such a list stores its elements in an array, but in arbitrary order. Additional pointers are used to link each element to the next smaller one in the list. Given a sorted compact list of size n, the smallest element can be found in expected time $O(\sqrt{n})$ as follows: sample a set of $\lfloor\sqrt{n}\rfloor$ array elements at random. Starting from their minimum, follow the predecessor pointers to the global minimum. The key fact is that the expected number of pointers to be followed is bounded by $\sqrt{n}$, and this yields the expected runtime.

On an abstract level, the situation can be modeled as follows. Let H be a set of size n that we can think of as the set of constraints in an optimization problem, for example the elements in a sorted compact list. Let $V : 2^H \to 2^H$ be a function that assigns to each subset $R \subseteq H$ of constraints a set $V(R) \subseteq H \setminus R$. We can think of $V(R)$ as the set of constraints violated by the optimal solution subject to only the constraints in R. In the sorted compact list example, $V(R)$ is the set of elements that are smaller than the minimum of R.

In this setting, the above "key fact" is a concrete answer to the following abstract question: Suppose that we sample a set $R \subseteq H$ of size $r \leq n$ uniformly at random. What can we say about the quantity v_r, the expected size of $V(R)$? What are conditions on V under which v_r is small?

The main workhorse in this context is the *Sampling Lemma* [6]. It states that $v_r = \frac{n-r}{r+1} \cdot x_{r+1}$, where x_r is the expected size of $X(R) = \{h \in R : h \in V(R\backslash\{h\})\}$. In other words, $h \in X(R)$ is a constraint that is not automatically satisfied if the problem is solved without enforcing it. In the sorted compact list example, every nonempty set R has one such "extreme" constraint, namely its minimum. Consequently, we have $x_{r+1} = 1$, and hence $v_r = (n-r)/(r+1)$. With $r = \lfloor\sqrt{n}\rfloor$, $v_r < \sqrt{n}$ follows. The Sampling Lemma has many other applications in computational geometry when x_{r+1} can be bounded [6].

In this paper, we address the following more general question in the abstract setting: Suppose that we sample a set $R \subseteq H$ of size $r \leq n$ uniformly at random, but then we remove a subset $K_R \subseteq R$ of a fixed size k, according to an arbitrary but fixed rule. What can we still say about the expected size of $V(R\setminus K_R)$? If K_R is a random subset of R, the expectation is v_{r-k}, but if K_R is chosen by another (deterministic) rule, then $R \setminus K_R$ is no longer a uniformly random subset, and the Sampling Lemma does not apply.

Our work is originally motivated by *chance-*

*Department of Computer Science, Institute of Theoretical Computer Science, ETH Zürich, CH-8092 Zürich, Switzerland, {gaertner, johannes.lengler, may.szedlak}@inf.ethz.ch.
Research supported by the Swiss National Science Foundation (SNF Project 200021_150055 / 1)

constrained optimization, see [6] and the explanations and references therein, but we also believe that the question is natural and interesting in itself.

A first bound on the change of the expected number of violated constraints was given in [3] in the case where (H, V) is a *nondegenerate LP-type problem*. The results are parametrized by the dimension δ (for definition of dimension see Definition 3 below). LP-type problems have been introduced and analyzed by Matoušek, Sharir and Welzl as a combinatorial framework that encompasses linear programming and other geometric optimization problems [9, 7]. The quantitative result was that under removal of k elements, the expected number of violated constraints increases by a factor of δ^k at most, which is constant if both δ and k are constant. It was left open whether this factor can be improved for interesting sample sizes (for very specific and rather irrelevant values of δ, r, k, it was shown to be best possible).

In this paper, we improve over the results in [3] in several respects. In Theorem 6 we show that the increase factor δ^k can be replaced by $\log n + k$, which is a vast improvement for a large range of values of k. Moreover, the new bound neither requires the machinery of LP-type problems, nor nondegeneracy. It holds in the completely abstract setting considered above. In this setting, we can also show that the bound is best possible for all sample sizes of the form $r = n^\alpha, 0 < \alpha < 1$. We also show that this bound is best possible for violator spaces, in the case where $k = \Omega(\delta \log n)$. In general, for violator spaces the gap to the lower bound is $\log n$.

Hence, if anything can be gained over the new bound, additional properties of the violator function V have to be used. Indeed, for small values of k, the increase factor in [3] is better than our new bound for nondegenerate LP-type problems, and most notably, it does not depend on the problem size n. We show in Theorem 9 that the same factor can be derived under the much weaker conditions of a *nondegenerate violator space*, and with a much simpler proof, based on a "removal version" of the Sampling Lemma (Lemma 8). Furthermore the proof of [3] is given for a specific rule to remove k, whereas our proof works for any rule.

Intuitively, violator spaces are LP-type problems without objective function, and they were introduced to show that many combinatorial properties of LP-type problems and algorithms for LP-type problems do not require the objective function at all [5, 1].

In Section 3, Theorem 10 we show tight upper and lower bounds for the case $\delta = 1$, which shows that the improved bound for nondegenerate violator spaces is best possible for *all* violator spaces. For smaller (and in particular constant) k, the quest for the best bound on the increase factor remains open.

What also remains open is the role of nondegeneracy. In many geometric situations, nondegeneracy can be attained through symbolic perturbation and can therefore be assumed without loss of generality for most purposes. In the abstract setting, this is not necessarily true, as there are examples of LP-type problems for which any "combinatorial perturbation" increases the dimension [8].

2 Basics and Definitions

Throughout the paper we will work with three combinatorial concepts, the LP-type problem, the violator space and the consistent space.

Definition 1 (LP-type Problems) *An* LP-type *problem is a triple $\mathcal{P} = (H, \Omega, \omega)$ that satisfies the following. H is a finite set (the constraints), Ω a totally ordered set with a smallest element $-\infty$ and $\omega : 2^H \to \Omega$ a function that assigns an objective function value to $G \subseteq H$, such that $\omega(\emptyset) = -\infty$. For all $F \subseteq G \subseteq H$ and $h \in H$, it holds that (1) $\omega(F) \leq \omega(G)$, and (2) if $\omega(F) = \omega(G) > -\infty$, then $\omega(G \cup \{h\}) > \omega(G) \Rightarrow \omega(F \cup \{h\}) > \omega(F)$. The first condition is called* monotonicity*, the second* locality*.*

A constraint $h \in H \setminus G$ is violated by G if $\omega(G \cup \{h\}) > \omega(G)$. We denote the set of violated constraints by $V(G)$. The classic example of an LP-type problem is the problem of computing the smallest enclosing ball (SEB) of a finite set of points P in $\mathbb{R}^d$ [10]. For SEB, the violated constraints of G are exactly the points lying outside the smallest enclosing ball of G.

Intuitively a violator space is an LP-type problem without an objective function.

Definition 2 (Violator Space) *A* violator space *is a pair (H, V), $|H| = n$ finite and $V : 2^H \to 2^H$ such that for all $F \subseteq G \subseteq H$, it holds that (1) $G \cap V(G) = \emptyset$ and (2) if $G \cap V(F) = \emptyset$, then $V(G) = V(F)$. The first condition is called* consistency*, the second* locality*.*

The notion of a violator space is more general than the LP-type problem, since every LP-type problem can naturally be converted into a violator space. On the other hand, not every violator space can be converted into an LP-type problem [5].

Definition 3 *Let (H, V) be a violator space.*

1. *$B \subseteq H$ is called a* basis *in (H, V), if for all $F \subsetneq B$, $B \cap V(F) \neq \emptyset$ (or equivalently, $V(F) \neq V(B)$).*
2. *A basis of $G \subseteq H$ is a basis B in (H, V) such that $B \subseteq G$ and $V(B) = V(G)$.*
3. *The* combinatorial dimension *of (H, V), denoted $\delta := \delta(H, V)$ is defined by the size of the largest basis in (H, V).*

For SEB, a basis of G is a minimal subset of points with the same enclosing ball of G. In particular, all points of the basis are on the ball's boundary. In d-dimensional space, the combinatorial dimension of any SEB-instance is at most $d+1$, since any enclosing ball can be defined by at most $d+1$ points on its boundary. However, a basis can be smaller than the combinatorial dimension, and a point set can have more than one basis: in $\mathbb{R}^2$ the set of four corners of a square has two bases, the two pairs of diagonally opposite points.

The set of *extreme constraints* $X(G) \subseteq G$ is defined by $r \in X(G) \Leftrightarrow r \in V(G \setminus \{r\})$.

In the SEB case, h is extreme in G if its removal allows for a smaller enclosing ball. Therefore h is necessarily on the boundary of the smallest enclosing ball, but this is not sufficient. For the case $\mathbb{R}^2$, if G consists of the four points on a circle, then G has no extreme point.

It is not hard to see that $X(G)$ is the intersection of all bases of G, hence $|X(G)| \leq \delta$. To bound the expected number of violators, the following result from [6] is known.

Lemma 4 *[Sampling Lemma] Let (H, V) be a violator space with combinatorial dimension δ. Let $R \subseteq H$ a u.a.r. set of size r, $v_r = \mathbb{E}[|V(R)|]$ and $x_r = \mathbb{E}[|X(R)|]$. Then $v_r = \frac{n-r}{r+1} \cdot x_{r+1} \leq \frac{n-r}{r+1} \cdot \delta$.*

The Sampling Lemma can be used to argue that v_r is small if the expected number x_{r+1} of extreme constraints of a random sample of size $r+1$ is small.

Hence in the SEB case every set has at most $d+1$ extreme points and therefore $v_r \leq \frac{n-r}{r+1} \cdot (d+1)$. If $d = 2$, then the smallest enclosing ball of a random sample of size $\sqrt{n}$ has in expectation at most $3\sqrt{n}$ points outside.

A violator space (H, V) is called *nondegenerate* if every set $G \subseteq H$ has a unique basis. Note that SEB it not nondegenerate, since as mentioned in $\mathbb{R}^2$, four points on a circle have two bases.

A consistent space is a violator space without the locality condition.

Definition 5 (Consistent Spaces) *A consistent space is a pair (H, V), $|H| = n$ finite and V a function $2^H \to 2^H$ such that for all $G \subseteq H$ it holds that $G \cap V(G) = \emptyset$.*

The basis, combinatorial dimension and extreme constraints of a consistent space can be defined equivalently as in the violator space.

In consistent spaces the first equality $v_r = \frac{n-r}{r+1} \cdot x_{r+1}$ of the Sampling Lemma 4 still holds. However, in general it does not hold that $|X(R)| \leq \delta$ for all $R \subseteq H$. One can construct examples where $X(R) = R$ [4].

3 Results

As already introduced in [3] for LP-type problems, we are interested in sampling with removal. We define the concept here for the most general case of consistent spaces. All results will then naturally extend for violator spaces and LP-type problems. Suppose we sample uniformly at random $R \subseteq H$ of size r. By some fixed rule P_k, we remove $k < r$ elements of R and obtain a set R_{P_k} of size $r - k$. We define $V_{P_k}(R) := V(R_{P_k})$. Note that in general (H, V_{P_k}) is not a consistent space. We are interested in $\mathbb{E}[|V_{P_k}(R)|]$, for which we will give several bounds. In Theorem 6 we give a tight bound for consistent spaces. In Theorem 9 we give a tight bound for nondegenerate violator spaces, which is an improvement to the result given in [3]. It depends on the values of δ and k whether the bound of Theorem 6 or Theorem 9 is stronger. Finally, in Theorem 10 we give a tight bound for violator spaces for the case where $\delta = 1$.

Tight Bounds on Consistent Spaces. The following result is proven by counting, the main argument is, that very few sets can have a large set of violators, i.e., $Pr[|V_{P_k}(R)| \geq x] \leq n^{-1}$ for x and n as defined below. For a full version of the proof see [4, Theorem 10].

Theorem 6 *Let (H, V), with $|H| = n$, a consistent space, δ, k, P_k and R with $|R| = r \leq n$ as above.*

$$\mathbb{E}[|V_{P_k}(R)|] \leq c \cdot \max\left\{\frac{n}{r}\delta \log n, \frac{n}{r}k\right\} =: x$$

where c is some suitable constant (e.g. $c = 33$).

For $\delta \log n = \Omega(k)$ tightness of the bound can be shown by choosing for every set of size at most δ, a set of violators of size $\Theta(\frac{n}{r}\delta \log n)$ independently and u.a.r. [4, Lemma 15]. For $\delta \log n = o(k)$ the bound is even tight for violator spaces [4, Lemma 17].

Extreme Constraints after Removal. Let (H, V) be a violator space of combinatorial dimension δ. In particular, every set has at most δ extreme constraints. For a given natural number k, we want to understand the following quantity:

$$\Delta_k(H, V) := \max_{R \subseteq X} |\{X(R \setminus K) : K \subseteq R, |K| = k\}|.$$

In other words, how many sets of extreme constraints can we get by removing k elements from some set R?

We obviously have $\Delta_0(H, V) = 1$ for any violator space (H, V). Moreover for (H, V) *nondegenerate* we have $\Delta_1(H, V) \leq \delta + 1$. Indeed one can show that if we remove a non-extreme element x from R, we end up with the same set $X(R \setminus \{x\}) = X(R)$ of extreme elements, so only in at most δ cases, we will get a

different set. Note that this does not hold in general [4]. Continuing with the same argument the following bound follows (for a full proof see [4]).

Lemma 7 *Let (H, V) be a nondegenerate violator space. Then $\Delta_k(H, V) \leq \sum_{i=0}^{k} \delta^i$.*

Sampling Lemma after Removal. Let (H, V) be a violator space. For $R \subseteq H$ and a natural number k, we define the following two quantities: $V_k(R) = \{x \in H \setminus R : x \in V(R \setminus K)$ for some $K \subseteq R, |K| = k\}$ and $X_k(R) = \{x \in R : x \in X(R \setminus K)$ for some $K \subseteq R, |K| = k\}$. Clearly, $V(R) = V_0(R)$ and $X(R) = X_0(R)$. Furthermore, we let $v_{r,k} = \mathbb{E}[V_k(R)]$ and similarly $x_{r,k} = \mathbb{E}[X_k(R)]$.

Lemma 8 *[Sampling Lemma after Removal]*

$$v_{r,k} = \frac{n-r}{r+1} x_{r+1,k}.$$

The proof goes like the one for the "normal" Sampling Lemma 4 [6]. The main idea is to define a bipartite graph on the vertex set $\binom{X}{r} \cup \binom{X}{r+1}$, where we connect R and $R \cup \{x\}$ with an edge if and only if $x \in V_k(R)$. By counting the outgoing edges on both sides the lemma follows [4]. Again this equality holds for consistent spaces as well.

Violators after Removal. For $R \subseteq H$, let K_R be the k-element set removed by P_k, i.e., $R_{P_k} = R \setminus K_R$. Then $E[|V_{P_k}(R)|] \leq v_{r,k} + k$. This follows since $v_{r,k}$ counts the expected number of violators in $H \setminus R$ that we can possibly get by removing *any* set of k elements and the removed elements in K_R can also be in $V(R_{P_k})$.

Theorem 9 *Let (H, V) be a nondegenerate violator space, δ, k, P_k and R with $|R| = r \leq n$ as above. Then*

$$\mathbb{E}[|V_{P_k}(R)|] \leq v_{r,k} + k \leq \sum_{i=1}^{k+1} \delta^i \cdot \frac{n-r}{r+1} + k.$$

Proof. By Lemma 8 it suffices to show that $|X_k(R)| \leq \sum_{i=1}^{k+1} \delta^i$. This holds, since by Lemma 7, at most $\sum_{i=0}^{k} \delta^i$ many *sets* of extreme elements can be obtained by removing k elements from R, and each of these sets has at most δ elements. □

By [3, Section 7.2], there exists an LP-type problem and a rule P_k, such that $|X_k(R)| = \Theta(\delta^{k+1})$, for $|R| = n-1$. However, the behavior of the bound is unknown for general r.

Combinatorial Dimension 1. In the case of violator spaces it is open whether (or when) the upper bound of Theorem 6 is tight for $k < \delta \log n$. In this case, there is a gap of up to $\log n$ between upper and lower bounds [4, Lemma 17]. For $k = 0$ we know a stronger upper bound of $O(\frac{n-r}{r+1}\delta)$ by the Sampling Lemma 4.

For the case $\delta = 1$ one can show that there exists only one class of violator spaces of dimension 1 [4, Lemma 21], namely the class of the *smallest number with repetitions* violator space, which is defined as follows: Let $|H| = n$ and H a multiset of $[n]$, i.e., every element of H is in $[n]$ and there might be repetitions. For $R \subseteq H$, let $V(R) = \{x \in H \mid x < \min_{i \in R} i\}$. Finally we require that either $V(\emptyset) = H$ or $V(\emptyset) = V(i)$ for some $i \in H$. In this setting one can prove that $\mathbb{E}[|V_{P_k}(R)|] = O(\frac{n}{r}k)$ and that this bound is tight [4, Theorem 18]. The theorem below follows immediately.

Theorem 10 *Let (H, V) be a violator space with dimension $\delta = 1$. Let k, P_k and R with $|R| = r \leq n$ as above. Then $\mathbb{E}[|V_{P_k}(R)|] = O(\frac{n}{r}k)$, and this bound is tight.*

Acknowledgments. The authors are grateful to Kenneth Clarkson and Emo Welzl for sharing important insights. Furthermore we thank Luis Barba for useful discussions.

References

[1] Y. Brise and B. Gärtner. Clarkson's algorithm for violator spaces. *Comp. Geom.*, 44(2):70 – 81, 2011.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA., 1990.

[3] B. Gärtner. Sampling with removal in LP-type problems. *Journal of Comp. Geom.*, 6(2):93–112, 2015.

[4] B. Gärtner, L. Lengler, and M. Szedlák. Random sampling with removal. Preprint arXiv:1512.04226.

[5] B. Gärtner, J. Matoušek, L. Rüst, and P. Škovroň. Violator spaces: Structure and algorithms. *Discrete Appl. Math.*, 156(11):2124–2141, 2008.

[6] B. Gärtner and E. Welzl. A simple sampling lemma: Analysis and applications in geometric optimization. *Discrete & Comp. Geom.*, 25(4):569–590, 2001.

[7] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[8] J. Matoušek. Removing degeneracy in LP-type problems revisited. *Discrete & Comp. Geom.*, 42(4):517–526, 2009.

[9] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. of STACS*, pages 569–579, 1992.

[10] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *Results and New Trends in Comp. Science*, pages 359–370. Springer-Verlag, 1991.

Characterizing the Distortion of Some Simple Euclidean Embeddings

Jonathan Lenchner * Krzysztof P. Onak* Donald R. Sheehy † Liu Yang *

Abstract

We consider two related families of problems. First we consider the embedding of finite point sets on a circle into one or more lines, or finite point sets on a sphere onto one or more planes. Next we consider the problem of embedding $N+1$ points from $\mathbb{R}^{K+1}$ into $\mathbb{R}^K$ where all but one of the $N+1$ points are in $\mathbb{R}^K$. Given such point sets, in the worst case, how much distortion must necessarily be incurred, by the best embedding?

1 Introduction

Various authors have studied the problem of minimizing the distortion of embedding points from one metric space into another metric space. In this work we consider two related families of problems. First we consider the embedding of finite point sets on a circle into one or more lines, or finite point sets on a sphere onto one or more planes. Next we consider the problem of embedding $N+1$ points from $\mathbb{R}^{K+1}$ into $\mathbb{R}^K$ where all but one of the $N+1$ points are in $\mathbb{R}^K$. Given such point sets, in the worst case, how much distortion must necessarily be incurred, by the best embedding? In the case of the $N+1$ points, how does the maximum distortion compare to the case where an unbounded number of points can lie outside any particular hyperplane of $\mathbb{R}^{K+1}$? Questions of this nature are important in many application areas, from data compression to machine learning.

Notation: Let Π be an embedding of one metric space, $\mathcal{M}_1$ into a second metric space, $\mathcal{M}_2$. Let $d_1(x,y)$ denote the distance between two points $x,y \in \mathcal{M}_1$ and let $d_2(x,y)$ denote the distance between two points $x,y \in \mathcal{M}_2$.

Definition: Let P be a finite point set in a metric space $\mathcal{M}_1$, and let $\Pi : P \to \mathcal{M}_2$ be a mapping (embedding) of P into $\mathcal{M}_2$. Then the **distortion** of the mapping Π, $\mathrm{Dist}(\Pi)$ is given by

$$\mathrm{Dist}(\Pi) = \max\left(\max_{x,y\in P} \frac{d_2(\Pi(x),\Pi(y))}{d_1(x,y)}, \max_{x,y\in P} \frac{d_1(x,y)}{d_2(\Pi(x),\Pi(y))}\right).$$

2 Background and Related Work

A fundamental reference that discusses the Lipschitz extension theorem of Kirszbraun (see next section) and the now classical Johnson-Lindenstrauss-Schechtman Lemmas is [4]. [1] and [2] study embedding metric spaces into a line, and into the two-dimensional plane. Our work is most closely related to [3], which discusses online metric embeddings. [5] and [6] are two older works that study the embedding of finite metric spaces into low dimensional Euclidean spaces.

3 Embeddings Points on a Circle into a Line and Points on a Sphere into a Plane

Definition: Call a set of N points on the unit sphere S^K **dense** if the radius of the largest empty cap is of size $O\left(\frac{1}{N^{1/K}}\right)$.

Unless otherwise stated, all metrics are assumed to be the Euclidean metric of the ambient spaces. Badiou et al. [2] showed that any embedding of N points on the sphere into a plane has distortion $O(\sqrt{N})$ and that a dense set of points on the unit sphere embeds into $\mathbb{R}^2$ with distortion $\Theta(\sqrt{N})$. The proof of the latter uses the Borsuk-Ulam Theorem together with Kirszbraun's Theorem [4], which says that a Liptschitz embedding of a subset of a Hilbert Space into another Hilbert Space can be extended to a Liptschitz embedding of the full space, with the same Liptschitz constant. The same arguments can be used to show that any embedding of N points on a circle into a line has distortion $O(N)$ and that a dense set of points on the unit circle embeds into the line with distortion that is $\Theta(N)$.

4 Embeddings Points on a Circle into Multiple Lines and Points on a Sphere into Multiple Planes

We first consider the problem of embedding points on a circle into two lines.

Lemma 1 *A set of N points on a circle can be embedded into two lines selected by the problem solver with distortion that is $O(\sqrt{N})$.*

Proof. Consider an origin-centered disk of unit radius and a set P of N points on the disk. To this set of points add their antipodal points $-P$. Among the points in $P \cup -P$ there is some pair of adjacent points p, p' that are $\Omega(\frac{1}{N})$ from one another. The antipodals to these points are also adjacent to one another with the same separation. Split the circle with a diametric cut that passes between p and p', and also between $-p$ and $-p'$.

*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, {lenchner,konak,yangli}@us.ibm.com

†University of Connecticut, Storrs, CT 06269 don.r.sheehy@gmail.com

Half of the points of $P \cup -P$ will be on one side of this cut and half on the other side. Without loss of generality suppose the cut is the line $y = 0$. Now consider the two lines $y = \frac{1}{\sqrt{N}}$ and $y = -\frac{1}{\sqrt{N}}$. Define an embedding Π of the points on the circle to the two lines as follows. Embed the points on the top half of the circle, say that in left to right order they are $p, ..., -p'$, onto $y = \frac{1}{\sqrt{N}}$, also in left to right order, so that their sequential distances from one another are the same as their geodetic distances on the circle. Then embed the points on the bottom half of the circle, i.e. $-p, ..., p'$, this time in their natural right to left ordering, onto $y = -\frac{1}{\sqrt{N}}$, so that their sequential distances from one another are again the same as their geodetic distances on the circle, and moreover, such that the image of $-p$ is directly below the image of $-p'$.

For points q, q' that are mapped to the same line, Π introduces just a constant amount of distortion since the geodetic distance along the circle is an $O(1)$-approximation to the Euclidean distance. To see this formally we need show that the ratio of the length of an arc of the unit circle to the associated chord length is bounded by a constant. On a unit circle the length of an arc associated with a central angle θ is also θ, while the associated chord length is $\sqrt{2-2\cos\theta} = 2\sin\frac{\theta}{2}$. So we must determine the supremum of

$$f(\theta) = \frac{\theta}{2\sin\frac{\theta}{2}}. \tag{1}$$

By L'Hôpital's Rule,

$$\lim_{\theta \to 0} f(\theta) = 1, \tag{2}$$

and

$$f'(\theta) = \frac{2\sin\frac{\theta}{2} - \theta\cos\frac{\theta}{2}}{4\sin^2\frac{\theta}{2}} \tag{3}$$

and the numerator is never 0. It follows that the supremum of f is the maximum of 1 (the effective value of $f(0)$) and $f(\pi) = \frac{\pi}{2}$, establishing that the geodetic distance along the circle is an $O(1)$-approximation to the Euclidean distance.

Thus the biggest distortion is either the distortion introduced at $\Pi(p), \Pi(p')$ (equivalently, at $\Pi(-p), \Pi(-p')$), which is potentially the most pronounced expansion, or by the most pronounced contraction, which can be no more substantial than if there were points at $(0,1), (0,-1)$ with $(0,1)$ embedding into the line $y = \frac{1}{\sqrt{N}}$ and $(0,-1)$ embedding into the line $y = -\frac{1}{\sqrt{N}}$.

However the distortion at $\Pi(p), \Pi(p')$ is

$$O\left(\frac{\frac{1}{\sqrt{N}}}{\frac{1}{N}}\right) = O(\sqrt{N}), \tag{4}$$

while the distortion assuming there were points at $(0,1), (0,-1)$ would be

$$O\left(\frac{1}{\frac{1}{\sqrt{N}}}\right) = O(\sqrt{N}). \tag{5}$$

Thus the lemma is established. □

Lemma 2 *A set of N points on a circle can be embedded into three lines selected by the problem solver with constant distortion.*

Proof. We consider the unit circle, C, and an associated circumscribed equilateral triangle, T. We map the N points on C to N geodetically proportionally spaced points on T, respecting the ordering of the points on C. Call this map Π. We show that Π can neither expand nor contract distances by too much. The proof that Π does not expand too much breaks down into a series of three fairly trivial observations, namely: (i) The geodesic distance on the circle is an $O(1)$-APX to the associated Euclidean distance, (ii) the geodesic distance on the triangle is an $O(1)$-APX to the geodesic distance on the circle, and (iii) the Euclidean distance between two points on a triangle is never greater than the geodesic distance on the triangle (obvious). That Π is at most a constant factor expansion means that the Euclidean distance on the triangle is at most a constant factor expansion to the associated Euclidean distance on the circle. The result follows by the transitivity of the $O(1)$-APX relation if we can establish (i) and (ii).

We established the truth of (i) in our proof of Lemma 1. (ii) is even easier since the approximation ratio is just the ratio of the associated perimeters, which is $\frac{3\sqrt{3}}{\pi}$. The fact that Π expands by at most a constant factor follows.

To show that Π contracts by at most a constant factor, it suffices that (a) the geodesic distance on the circle does not decrease distances relative to the Euclidean distance on the circle, (b) the geodesic distance on the triangle does not decrease distances relative to the geodesic distance on the circle, and (c) the Euclidean distance on the triangle does not contract distances by more than a constant factor relative to the geodesic distance on the triangle. (a) and (b) are obvious. For (c) consider Figure 1. By the law of

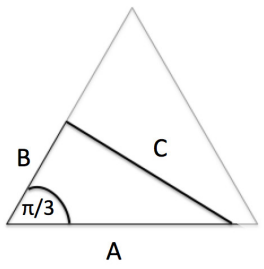


Figure 1. Comparison of the Euclidean distance, C, between two points on an equilateral triangle, and the geodesic distance $A+B$.

cosines,

$$C^2 = A^2 + B^2 - 2AB\cos\frac{\pi}{3} = A^2 + B^2 - AB. \tag{6}$$

Now, without loss of generality, assume that $A \geq B$ and that $A = m + \varepsilon$, $B = m - \varepsilon$ (where $m = \frac{A+B}{2}$ and $\varepsilon = \frac{A-B}{2}$). Then

$$\begin{aligned} C^2 &= (m+\varepsilon)^2 + (m-\varepsilon)^2 + (m+\varepsilon)(m-\varepsilon) \\ &= m^2 + 3\varepsilon^2, \end{aligned}$$

and so $C \geq \frac{A+B}{2}$. Thus the Euclidean distance contracts by no more than a factor of 2 relative to the geodesic distance on the equilateral triangle and so Π contracts by at most a constant factor, and the lemma is established. □

We observe that it is not possible to extend Lemma 2 to an annulus of constant thickness. If it were possible to make such an extension then it would be possible to embed points on an ε-thick annulus into a disk with constant distortion. However, in what is a variant of a rather usual argument/counterexample, consider N points in the annulus contained within a $\sqrt{N}$ x $\sqrt{N}$ square grid, each grid point at a distance of $\delta = \varepsilon/N$ from the next. If we embed these points onto the circle so that the distortion of each point with its neighbor on the circle gets constant distortion, then they must be placed at distance no smaller than $k\delta$ from one another from some constant k. However, the two furthest apart points on the circle will be approximately $Nk\delta$ from one another, while they started at distance no greater than $\sqrt{2N}\delta$ from one another. Thus they incur a distortion of at least $\sqrt{N}$.

It is conceivable, however, that Lemma 1 can be extended to cover the case of an annulus of constant thickness.

Definition: Say that a set of N points on the sphere is distributed **approximately uniformly** if the geodetic distance between any two points in the set is $\Omega(\frac{1}{\sqrt{N}})$ and there is no empty patch (cap) of radius $\Omega(\frac{1}{\sqrt{N}})$.

We believe the next lemma holds for an arbitrary set of N points on the sphere but the best we can prove at present is the following:

Lemma 3 *An approximately uniformly distributed set of N points on a sphere can be embedded into two planes selected by the problem solver with distortion that is $O(N^{1/4})$.*

Proof. Place N points approximately uniformly on a unit radius sphere. We will embed the points on the surface of the sphere onto two planes at $z = \pm\frac{1}{N^{1/4}}$. Points on the bottom half of he sphere will be embedded onto the $z = -\frac{1}{N^{1/4}}$ plane via the following two step process: (1) Project each such point p first to the $z = -1$ plane via the unique line through p that makes a 45° angle with the z-axis. Map the south pole to the south pole. (2) Map points from the $z = -1$ plane to the $z = -\frac{1}{N^{1/4}}$ via vertical projection. Do analogously to embed points on the northern hemisphere into the $z = \frac{1}{N^{1/4}}$ plane.

As in the case of the circle, projection of points on a common hemisphere onto a plane incurs a constant amount of distortion. For points near the equator and near the south pole there is essentially no distortion while the distortion is maximized for points midway between the equator and the south pole, when the distortion is easily seen to be $\sqrt{2}$.

Thus the biggest possible distortion either arises at the mapping of potential points (p_N, p_S) where p_N denotes a point at the north (top) pole of the sphere and p_S denotes its antipodal point, or at pairs of points $(p_{(\theta,\phi)_{\text{top}}}, p_{(\theta',\phi')_{\text{bottom}}})$, which denote a pair of points initially as close as possible but on opposite hemispheres, and which therefore get embedded into different planes. The distortion in the case of (p_N, p_S) is:

$$\text{Dist} = \frac{2}{\frac{2}{N^{1/4}}} = N^{1/4}. \tag{7}$$

While the distortion at $(p_{(\theta,\phi)_{\text{top}}}, p_{(\theta',\phi')_{\text{bottom}}})$ is

$$\text{Dist} = \frac{\frac{2}{N^{1/4}}}{O(\frac{1}{\sqrt{N}})} = O(N^{1/4}), \tag{8}$$

establishing the lemma. □

Lemma 4 *Any set of N points on the surface of a sphere can be embedded into four planes selected by the problem solver with constant distortion.*

The proof of this lemma proceeds by projecting the N points on the sphere outward onto the regular tetrahedron that has the given sphere as its inscribed sphere. One then verifies that the Euclidean distance between two projected points is both bounded above and below by a constant factor times the Euclidean distance determined by the original points. The proof, the details of which we omit, is similar in spirit, though a bit more cumbersome than the proof of Lemma 2.

5 Embedding N Points on a Line and One Point off the Line onto a Line or N Points on a Plane and One Point off the Plane onto a Plane

Lemma 5 *Consider a collection of an odd number, N, of points on a line, each point one unit from the next, together with one additional point at height $\sqrt{N}$ above the center point of the points on the line. Then any embedding of these points into a line has distortion $\Omega(\sqrt{N})$*

Proof (sketch). Label the points consecutively along the line by $P = \{p_1, ..., p_N\}$, and refer to the point above the line at distance $\sqrt{N}$ by q. Further, denote the central point among the points in P, $p_{\frac{N+1}{2}}$, by p_{cent}.

We prove the lemma by contradiction. Suppose we have a non-contracting embedding Π of $P \cup \{q\}$ into a line, which has distortion $o(\sqrt{N})$. Consider first that some of the points in P are mapped under Π to one side of $\Pi(q)$ and some to the other. It must then be the case that some pair of adjacent points p_i and p_{i+1} are mapped by Π to opposite sides of $\Pi(q)$. But if p_i is mapped to one side of $\Pi(q)$ and p_{i+1} is mapped to the other, then, by non-contraction, the distance between $\Pi(q)$ and each of its closest neighbors is at least $\sqrt{N}$ and thus $d(\Pi(p_i), \Pi(p_{i+1})) \geq 2\sqrt{N}$ so the distortion in Π is at least $2\sqrt{N}$. Thus, for Π to have distortion $o(\sqrt{N})$ all points $\Pi(p_i)$ must be to one side of $\Pi(q)$.

Since all points $\Pi(p_i)$ are to one side of $\Pi(q)$ there is a closest neighbor $\Pi(p^*)$ to $\Pi(q)$. Divide the points of P, as evenly as possible into four sequential quarters. p^* either comes from one of the outer quarters of $\{p_1,...,p_N\}$ or from one of the two inner quarters. Label these quarters $P_{1/4}, P_{2/4}, P_{3/4}$ and $P_{4/4}$, respectively.

On the one hand, if $p^* \in P_{1/4} \cup P_{4/4}$, a calculation shows that the distortion in the mapping of q, p_{cent} is at least $\frac{\sqrt{N}}{2}$.

On the other hand, if $p^* \in P_{2/4} \cup P_{3/4}$, an analogous computation shows that there must be a pair of consecutive points p_i, p_{i+1} whose distortion is at least $N/2$, establishing the lemma.

Lemma 6 *Consider a set of N points inside a disk of radius $\sqrt{N}$ with largest empty subdisk of size $O(1)$, together with one additional point at height $N^{1/4}$ above the center point of the points in the disk. Then any embedding of these points into the plane has distortion $\Omega(N^{1/4})$.*

Proof (sketch). Suppose we have a non-expanding embedding Π of the N points, P, in the disk, together with the point above the center of the disk, which we again call q, into the plane. Extend Π to be a non-expanding embedding of all of $\mathbb{R}^3$ into $\mathbb{R}^2$ by Kirszbraun's Theorem. Since Π is Lipschitz (with Lipschitz constant at most 1), Π is continuous. Let p_{cent} be the centerpoint in P directly below q. Consider the image under Π of the vertical diameter of the disk $\Pi(\text{diam})$. This image is a continuous curve through $\Pi(p_{\text{cent}})$. Color the top half of $\Pi(\text{diam})$ red and the bottom half green. Now consider the image $\Pi(\text{diam})$ as the diameter turns through 180 degrees. Continue to color $\Pi(\text{top-half})$ red and $\Pi(\text{bottom-half})$ green. A straight forward argument shows that either the endpoints of the red and green halves of these curves collectively form a closed curve with $\Pi(p_{\text{cent}})$ in its interior *or* at some point in the turning of the diameter either the end point of the green curve intersects the red curve or the end point of the red curve intersects the green curve. Suppose one of these latter two cases holds, say it is that the end point of the red curve intersects the green curve. If p_{r_e} is the pre-image of the end point of the red curve at this juncture, then there is a point p_g which is the pre-image of a point along the green curve such that $d(\Pi(p_{r_e}), \Pi(p_g)) \approx 1$ while the points p_{r_e}, p_g lie along a diameter and are at least $\sqrt{N}$ apart in the pre-image. Thus, in this case, $Dist(\Pi) = \Omega(\sqrt{N})$.

On the other hand, if $\Pi(p_{\text{cent}})$ is in the interior of the image of the disk then consider $\Pi(q)$, the image of the point above p_{cent}. If $\Pi(q)$ lies inside the image of the boundary of the disk, then since Π is non-expanding there is a point of the disk that is approximately distance 1 or less from $\Pi(q)$. Since the point started at least at distance $N^{1/4}$, the incurred distortion is $\Omega(N^{1/4})$. On the other hand, if the boundary of the disk lies between $\Pi(q)$ and $\Pi(p_{\text{cent}})$ then we again find a distortion of $\Omega(N^{1/4})$.

Future Work

These results are just the first of a hoped for more detailed characterization of how one incurs distortion on a point-by-point basis embedding from one Euclidean space into another of smaller dimension. In general if N points in $\mathbb{R}^k$ can incur some maximum distortion when the points are embedded in $\mathbb{R}^{k'}$, for $k' < k$, how much distortion can be incurred from a point set of the same size N, but where all but M of the N points lie in some k'-flat, and $M = o(N)$?

Many questions also remain regarding the low distortion embedding of points on an N-sphere into hyperplanes. For the 2-sphere we have results for one, two and four planes selected by the problem solver, but how about three planes? Can one achieve lesser order of magnitude distortion using three planes than two? We currently do not know how to do this and speculate that it is not possible.

References

[1] M. Bădoiu, J. Chuzhoy, P. Indyk, and A. Sidiropoulos. Low-distortion embeddings of general metrics into the line. In *In STOC05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 225–233. ACM, 2005.

[2] M. Bădoiu, K. Dhamdhere, A. Gupta, Y. Rabinovich, H. Räcke, R. Ravi, and A. Sidiropoulos. Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 119–128, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[3] P. Indyk, A. Magen, A. Sidiropoulos, and A. Zouzias. Online embeddings. In M. Serna, R. Shaltiel, K. Jansen, and J. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 6302 of *Lecture Notes in Computer Science*, pages 246–259. Springer Berlin Heidelberg, 2010.

[4] J. Lee and A. Naor. Absolute Lipschitz extendability. In *C. R. Acad. Sci. Paris*, volume 338, pages 859–862, 2003.

[5] R. Mathar. The best Euclidian fit to a given distance matrix in prescribed dimensions. *Linear Algebra and its Applications*, 67:1 – 6, 1985.

[6] J. Matoušek. Bi-Lipschitz embeddings into low-dimensional Euclidean spaces. *Commentationes Mathematicae Universitatis Carolinae*, 031(3):589–600, 1990.

A New Modular Parametric Search Framework

Christian Knauer* David Kübel† Fabian Stehn*

Abstract

Parametric search is a technique to develop efficient deterministic algorithms for optimization problems. The strategy requires an algorithm for the decision variant of the problem at hand. Given this decision algorithm, the strategy can be seen as a *black box* that does not really exploit characteristics of the underlying problem: It computes an *optimal* solution by keeping track of the values that appear in comparisons of the decision algorithm.

In this abstract, we present a new parametric search framework written in `Java`. We show how parametric search based algorithms can be implemented and explored with the framework. It allows to quickly specify or change crucial functionalities which influence the specific behaviour (and performance) of the resulting optimization algorithm. We focus on a transparent, simple-to-use, and modular design, and discuss the implementation of a specific algorithm that computes an optimal shortcut of a polygonal curve.

1 Introduction

Parametric search is a powerful and fairly general method to compute the (exact) optimal solution λ_{opt} of an optimization problem P in one variable. Let P be a minimization problem whose objective function is monotone, that is, for the decision variant DEC_P of P there is a value λ_{opt} such that $\text{DEC}_P(\lambda) = \text{TRUE} \Leftrightarrow \lambda \geq \lambda_{opt}$. In oder to apply Megiddo's [6] parametric search technique, two ingredients are required:

1. A sequential algorithm $\mathcal{A}_s$ that solves DEC_P for any value λ in T_s time, and
2. a parallel algorithm $\mathcal{A}_p$ using k processes, solving DEC_P for an unknown value of λ_{opt} in T_p time.

$\mathcal{A}_p$ is an algorithm that uses k (independent) processes whose control-flow depends on the outcome of comparisons. Each comparison can be resolved by relating a value that is derived from the input to the (unknown) value of λ_{opt}. The basic idea is to follow the control-flow of each process until it requires the solution to such a comparison and to collect these comparisons in *batches* (these collected values are usually called *critical values*). A batch hence consists of k values, representing k comparisons to λ_{opt}. Exploiting the monotonicity of P, all comparisons of a batch can be resolved by $O(\log k)$ calls to $\mathcal{A}_s$. After resolving a batch of critical values, each process can continue until a new batch of k comparisons is collected and resolved in the same way.

Throughout this process, an interval $I = (\lambda^-, \lambda^+]$ is maintained, where λ^- is the largest value encountered so far such that $\text{DEC}_P(\lambda^-) = \text{FALSE}$ and λ^+ is the smallest value encountered so far with $\text{DEC}_P(\lambda^+) = \text{TRUE}$; this implies that $\lambda_{opt} \in I$. Through the course of this process, at least one instance of the parallel algorithm $\mathcal{A}_p$ has to carry out a comparison with the actual optimal value λ_{opt}, which implies that λ_{opt} will appear as a critical value. Since P is a minimization problem, we have that $\lambda_{opt} = \lambda^+$ after all processes of $\mathcal{A}_p$ terminated. This gives a combined minimization algorithm $\mathcal{C}$ that computes λ_{opt} in $O(T_p \cdot (k + T_s \cdot \log k))$ deterministic time; see [6] for details.

1.1 Applications

The technique has been applied to a wide range of problems. In the field of computational geometry, e.g., the technique has been used to compute the Fréchet Distance [2] between two polygonal curves. Recently, Große et al. [4] showed how to efficiently compute a diameter-optimal shortcut between vertices of a polygonal curve. Agarwal et al. [1] present several applications in context of geometric optimization.

1.2 Related work

The first implementations of parametric search algorithms are due to Toledo [9] (solving extremal polygon placement problems) and Schwerdt et al. [8] (computing the diameter of moving points) roughly twenty years ago. Van Oostrum and Veltkamp [7] were the first to provide a general framework, written in `C++`. Their framework includes a detailed documentation as well as two reference implementations to compute the *Median of Lines* [6] and the *Fréchet distance between two polygonal curves* [2]. They experimented with different sorting algorithms and showed that Quicksort can replace a parallel sorting network in practice.

*Institut für Informatik, Universität Bayreuth, {christian.knauer, fabian.stehn}@uni-bayreuth.de,

†Institut für Informatik, Abteilung I, Universität Bonn, dkuebel@uni-bonn.de

1.3 Contribution

In Chapter 2, we present a new general parametric search framework. Two (geometric) algorithms that have been realized in this framework are discussed in Chapter 3.

The framework can be used as a black box to implement efficient optimization algorithms (based on decision algorithms) with minimal adaptation effort for the task at hand. It is designed to provide the capability to easily exchange not only the sorting algorithm (a central component of the parametric search technique) but also the strategy that is used to solve the comparisons of a batch, or the scheduler that manages the (simulated) parallel execution of the individual processes. Standard performance enhancements, such as "Cole's trick" [3] are built in along with other optimizations. Several parameters can be used to fine-tune the performance of the final algorithm.

From this point of view, our approach is twofold: On the one hand, the framework can be used as a black box that merely requires to provide an implementation of the corresponding decision algorithm and parts of $\mathcal{A}_p$ that determine and organize critical values. On the other hand, it allows a deeper look "under the hood" of the general mechanism of parametric search in order to study, analyse and compare an algorithm and to gain deeper insight into the original problem.

In Chapter 3, we discuss how a recent algorithm for computing shortcuts has been realized with the framework, and we show how the framework allows to study the effect of optimization strategies on the actual computation times. Numerical effects which may arise in practical applications are discussed briefly.

2 The Framework

In this section, we describe the general structure of our framework and discuss where and why it differs from the reference framework by van Oostrum et al. [7]. To keep the implementation effort as small as possible, van Oostrum et al. identify essential parts of the technique so that certain components can be reused in different implementations. Their framework provides Quicksort and Bitonicsort together with an automated organization of the comparisons in a batch. The framework encapsulates and hides the scheduling of comparisons and the management of critical values. In case that $\mathcal{A}_p$ is a parallel sorting algorithm, it is certainly a benefit to hide all this complexity which reduces the implementation effort considerably. If, however, $\mathcal{A}_p$ is not a sorting algorithm, the developer has to use "lower-level facilities for batching comparisons and suspending/resuming computation" instead which enforces the use of a rigid mechanism specified by their framework. This has certain drawbacks: It forces the developer to scatter code to scheduled methods which makes the parallel algorithm even harder to read, understand or debug. Realizing complex parallel algorithms is considerably more involved with this design. Moreover, the developer has no chance to control or influence the parallel steps and the evaluation of batches separately.

With our framework we offer the chance to hook into the scheduling or the handling of critical values, if desired or necessary. Both frameworks share (conceptually) similar components, e.g., a scheduler, processes or a decision algorithm. However, the design of our framework meets additional requirements. The most striking difference is that we separate the scheduling component from the management of critical values. We provide ready-to-use functionalities to reduce the implementation effort; c.f. Section 2.2. Among these are different strategies to resolve critical values of a batch which can be easily exchanged to study their impact on the overall performance. This provides an insight into how critical values are processed in a certain application and may reveal characteristics of the underlying problem.

The core components of the framework have already been released online [10]. The code of the whole parametric search framework together with the demo applications will be released as a part of a larger framework within this year.

2.1 Design Choices

Our framework is written in `Java` and consists of four components. A single responsibility is assigned to each component in order to (re-)use or interchange them independently. In the following, we briefly describe these components and their role within the framework.

One component is the serial decision algorithm $\mathcal{A}_s$ which has to be provided by the developer. The implementation has to support a method to decide DEC_P for any given value $\lambda \geq 0$. The outcome of a call to $\mathcal{A}_s$ is a boolean value; either TRUE (if $\lambda \geq \lambda_{opt}$) or FALSE (if $\lambda < \lambda_{opt}$).

The remaining three components constitute the parallel algorithm $\mathcal{A}_p$. We aim to restrict the access to the decision algorithm during the execution of $\mathcal{A}_p$. Whenever DEC_P needs to be solved for a concrete value λ, the component `Oracle` has to be asked: In contrast to $\mathcal{A}_s$, the oracle may also return the value UNKNOWN, implying that $\lambda \in I$, which forces the calling process to wait. This allows us to delay a single evaluation of the decision problem and to batch several critical values. Of course, at some points during the execution of $\mathcal{A}_p$ it will be necessary to evaluate the decision problem for (some of) the batched values, e.g., when $\mathcal{A}_p$ has to continue with its next parallel step. At this point, the oracle will apply a strategy

to resolve the comparisons of the current batch.

The flow of control of the parallel algorithm branches at certain points. This is realized by instances of `Process`, which allows for pausing and resuming, depending on the outcome to the calls to the oracle.

The component that manages the (virtual) parallel execution of $\mathcal{A}_p$ is the `Scheduler`: It assures that the oracle and all processes are triggered when necessary. After a certain number of parallel steps, all processes will terminate; as soon as the last processes becomes inactive, the scheduler and with it the entire algorithm will terminate.

2.2 Functionality and Oracle Strategies

With the framework we provide a parallel sorting algorithm based on a bitonic sorting network. It can be used to implement sorting based parametric search algorithms. The `Scheduler` is realized by a simple serial round-robin strategy. To experiment with different strategies of the `Oracle` component, we implemented the following strategies:

1. *Brute force.* This strategy does not store λ and solves the decision problem, at once. We do not maintain I. Consequently every request causes an evaluation of $\mathcal{A}_s$.
2. *Monotonicity.* This strategy only exploits the monotonicity of DEC_P: The decision problem is only evaluated if $\lambda \in I$. Otherwise, the outcome is determined according to the position of λ according to I in constant time.
3. *Parametric Search.* If λ lies in I, we store it in a list and return UNKNOWN. When the oracle is triggered via the method *resolveCollectedValues*, all stored values are resolved in a binary search fashion as suggested by Megiddo [6].
4. *Cole (unweighted).* In contrast to the previous strategy, the oracle evaluates DEC_P only for the median the of stored values. Afterwards, half of the critical values can be resolved in constant time. The corresponding processes are called to produce additional critical values.
5. *Cole (weighted).* To guarantee that no processes has to wait for a result for too long, a critical value is stored together with a certain weight. In contrast to the previous strategy, the decision problem is now evaluated for the weighted half of the stored values and not just for the median.

Depending on the specific application at hand, other strategies to handle and resolve batches can be realized or combined with these strategies. As a measurement for the performance we suggest to look at the total number of evaluations of $\mathcal{A}_s$.

3 Computing Optimal Shortcuts

In this section we briefly discuss two proof-of-concept implementations. Due to space limitations, we merely state the first implementation, the computation of the Fréchet Distance between two polygonal curves (c.f. [2]). For this implementation, Bitonicsort has been chosen to realize the parallel part of the algorithm; the corresponding decision problem was solved in a standard fashion via the use of free space diagrams.

The initial motivation that led to the design of this new framework is the problem of computing a *diameter-optimal shortcut* of a polygonal curve: A shortcut (a non-edge between two vertices of the path) is considered diameter-optimal if no other shortcut added to the curve results in a graph with smaller diameter. Große et al. [4] present a parametric search algorithm for this problem. In contrast to the computation of the Fréchet distance, their approach does not involve a sorting algorithm for $\mathcal{A}_p$.

Some Details

Let λ_{opt} denote the diameter induced by an optimal shortcut. The main idea of the concrete decision algorithm $\mathcal{A}_s$ is to check for every possible start vertex s of the shortcut whether there exists a feasible end vertex e. The vertex range of candidates for e is efficiently restricted through binary searches. $\mathcal{A}_s$ returns TRUE exactly if a feasible pair (s,e) was found.

Große et al. suggest to implement the decision algorithm in a generic and parallel fashion to get $\mathcal{A}_p$. This implies that each comparison in a binary search has to be deduced from the result of the decision problem at a critical value. For every candidate start vertex s, the search for a feasible e is independent and can be assigned to a separate parallel process. Consequently, Cole's weighting scheme [3] can be applied to reduce the theoretical worst-case running time by a log-factor.

We generated 1000 polygonal curves of 2^{10} vertices each, where the coordinates of the vertices were chosen uniformly at random within a square. For each curve, we computed the diameter-optimal shortcut using each of the five oracle strategies discussed above. As stated earlier, the number of calls to $\mathcal{A}_s$ is used to measure the performance of the individual strategies. Table 1 lists the outcome of the experiments, ordered by oracle strategy as discussed in Section 2.2. As expected, all strategies but the first (brute force), require a small number of evaluations of $\mathcal{A}_s$. With regard to the average number of calls, Strategy 2 performs best by only exploiting the monotonicity of the decision problem. As the last row of Table 1 reveals, Strategy 5, known as "Cole's trick" (which reduces the theoretical worst-case runtime by a log-factor), requires more calls to the decision problem than Strate-

	Strategy of the oracle				
	1	2	3	4	5
μ	$\sim$ 13,872	19.43	19.66	20.32	20.05
σ	$\sim$ 658	3.88	2.28	1.94	2.96
max	18,538	36	26	25	59
min	12,965	12	12	12	12

Table 1: Experimental results by oracle strategy. μ: average number of calls to $\mathcal{A}_s$; σ standard deviation; **max** (**min**): maximum (minimum) number of calls to $\mathcal{A}_s$.

gies $2-4$ for some instances.

The fact that Strategy 2 performs well is probably due to the order in which critical values are computed by $\mathcal{A}_p$. All distances from the first vertex to other vertices along the curve are critical values of the first batch (see [4] for details). Consequently, the interval I is already narrowed down right after the first batch has been resolved. The higher number of evaluations for Cole's weighing scheme in some instances might be due to the fact that this strategy calls $\mathcal{A}_s$ for values of λ that are far from λ_{opt}, as critical values of previous parallel steps are favoured over critical values of processes that have proceeded further.

It turns out that for the problem of computing an optimal shortcut, the critical values of the first batch are the key to achieve a small overall number of evaluations. The order in which critical values are computed and resolved plays an important role for the actual performance of a parametric search algorithm.

3.1 Numerical Issues

As for the built-in components of our framework, critical values are currently represented as double-precision numbers. The following problem can arise when representing critical values as finite-precision floats: The floating point representation of the optimal solution λ_{opt} is slightly smaller than the actual value of λ_{opt}. As a consequence, $\mathcal{A}_s$ rejects this value and it will be stored as the lower bound of $I = (\lambda^-,\lambda^+]$. If the algorithm outputs λ^+ as suggested above, the error can be huge. To address this problem, we can perform a final call to $\mathcal{A}_s$ with the center of I. If the center is a valid solution, we conclude that λ^- is closer to λ_{opt} than λ^+.

We are currently working on a solution to integrate and provide types that allow comparisons with arbitrary precision. Note that the general framework does not depend on specific numerical data-types. The specification and treatment of critical values has to be handled by the developer in a concrete application.

4 Conclusion & future work

With the presented framework it is possible to take a closer look into the details and the specific behaviour of parametric search algorithms. Different oracle strategies allow for quantified experiments under different optimization variants with no additional implementation effort. The outcome of these experiments might lead to a deeper understanding of the structure of the underlying problem.

Future versions of the framework might include the option to trace and visualize critical values throughout the parallel steps. In case of the shortcut problem, this would help to study the distribution of critical values around λ_{opt} and to understand under which conditions which parallel step calls $\mathcal{A}_s$ with the optimal value.

References

[1] Pankaj K. Agarwal, Micha Sharir and Sivan Toledo. *Applications of Parametric Searching in Geometric Optimization*, J. of Algorithms 17(3):292–318, 1994.

[2] Helmut Alt and Michael Godau. *Computing the Fréchet distance between two polygonal curves.* Int. J. of Comp. Geom. & App. 5:75–91, 1995.

[3] Richard Cole. *Slowing down sorting networks to obtain faster sorting algorithms.* J. ACM 34(1):200–208, 1987.

[4] Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel H. M. Smid and Fabian Stehn. *Fast Algorithms for Diameter-Optimally Augmenting Paths.* Proceedings, Part I, ICALP (1) 2015: 678–688.

[5] Prosenjit Gupta, Ravi Janardan and Michiel H. M. Smid. *Fast Algorithms for Collision and Proximity Problems Involving Moving Geometric Objects.* Comput. Geom. 6: 371-391, 1996.

[6] Nimrod Megiddo. *Applying Parallel Computation Algorithms in the Design of Serial Algorithms.* J. ACM 30(4): 852–865, 1983.

[7] René van Oostrum and Remco C. Veltkamp. *Parametric search made practical.* Comput. Geom. 28(2-3): 75–88, 2004.

[8] Jörg Schwerdt, Michiel H. M. Smid and Stefan Schirra. *Computing the Minimum Diameter for Moving Points: An Exact Implementation Using Parametric Search.* Proceedings of the Thirteenth Annual Symposium on Computational Geometry, pages 466-468, 1997.

[9] Sivan Toledo. *Extremal Polygon Containment Problems and other issues in parametric searching.* Master's Thesis, Tel-Aviv University, 1991.

[10] Source Code: *Modular Parametric Search Framework (in Java)* https://davidkuebel@bitbucket.org/-davidkuebel/modularparametricsearchframework.git

Detecting affine equivalences of planar rational curves

Michael Hauer and Bert Jüttler*

Abstract

We derive a system of polynomial equations to decide whether two rational parametric curves in the plane are related by an affine transformation and to detect all such affine equivalences. In order to do so, we use homogenization in both the parameter domain and the Euclidean plane. Furthermore, employing barycentric coordinates leads to a simple method for detecting affine equivalences, as these coordinates are invariant under affine transformations. In addition we interpret the result by relating the monomial coefficients to Bézier control points. Finally we provide numerical examples.

1 Introduction

Detecting symmetries is an essential problem in Pattern Recognition, Computer Graphics and Computer Vision. First approaches concentrated on point sets as input data. In 2004, Braß and Knauer [5] proposed to apply a point-based method to control polygons of Bézier curves and surfaces. For matching planar curve segments in B-spline form, a method based on affinely invariant moments has been described in [6]. Sánchez-Reyes [8] recently developed a method for symmetry detection of curves given in Bernstein-Bézier representation. Lebmeir and Richter-Gebert [7] investigated symmetries of algebraic curves given in implicit form. During the last two years, Alcázar et al. [1, 2, 3, 4] published a series of papers dealing with the problem of symmetry detection for parametric rational curves. They use the fact that the symmetry of a curve in proper parameterization can be related to a rational linear transformation in the parameter domain, see [9].

We consider properly parameterized rational curves and investigate the more general concept of affine equivalences. Symmetry detection can then be seen as a special case.

2 Detecting equivalences

Before presenting our method, we recall some geometric tools and clarify our notation.

*Institute of Applied Geometry, Johannes Kepler University of Linz, michael.hauer@jku.at, bert.juettler@jku.at

2.1 Preliminaries

We consider curves in the projectively closed Euclidean plane $\bar{E}^2$, whose points are given by homogeneous coordinate vectors $\mathbf{x} = (x_0, x_1, x_2)^T \in \mathbb{R}^3 \backslash \{(0,0,0)\}$. If there exists a $\mu \neq 0$, such that $\mathbf{x} = \mu\mathbf{y}$, $\mathbf{x}$ and $\mathbf{y}$ represent the same point in $\bar{E}^2$. We denote this by $\mathbf{x} \simeq \mathbf{y}$.

Three non-collinear base points $\mathbf{v}_0$, $\mathbf{v}_1$ and $\mathbf{v}_2$, none of which is a point at infinity, define a barycentric coordinate system, such that any finite point $\mathbf{x}$ possesses unique barycentric coordinates $\lambda_i(\mathbf{x})$, $i = 0, \dots, 2$, with respect to the base points. More precisely, we have

$$\frac{1}{x_0}\mathbf{x} = \sum_{i=0}^{2} \lambda_i(\mathbf{x}) \frac{1}{v_{i,0}} \mathbf{v}_i.$$

The barycentric coordinates can be computed using homogeneous coordinates

$$\begin{aligned} \lambda_0(\mathbf{x}; \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2) &= \Lambda(\mathbf{x}; \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2), \\ \lambda_1(\mathbf{x}; \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2) &= \Lambda(\mathbf{x}; \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_0), \\ \lambda_2(\mathbf{x}; \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2) &= \Lambda(\mathbf{x}; \mathbf{v}_2, \mathbf{v}_0, \mathbf{v}_1) \end{aligned}$$

where

$$\Lambda(\mathbf{x}; \mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{a_0 \det(\mathbf{x}, \mathbf{b}, \mathbf{c})}{x_0 \det(\mathbf{a}, \mathbf{b}, \mathbf{c})}. \tag{1}$$

Throughout the paper we consider two parametric rational curves $\mathcal{C}$ (and $\mathcal{C}'$, respectively) $\subset \bar{E}^2$, which are considered as point sets. Both curves are given by proper parameterizations[1]

$$\begin{aligned} \mathbf{p} : P^1(\mathbb{R}) &\to \mathcal{C} \subset \bar{E}^2, \\ \mathbf{t} \mapsto \mathbf{p}(\mathbf{t}) &= (p_0(t_0, t_1), p_1(t_0, t_1), p_2(t_0, t_1)) \,. \end{aligned}$$

The parameter $\mathbf{t} = (t_0, t_1)$ is a point on the projective line $P^1(\mathbb{R})$.

The homogeneous coordinates of the curves are homogeneous polynomials of degree n,

$$p_j(\mathbf{t}) = \sum_{i=0}^{n} c_{j,i} t_0^{n-i} t_1^i$$

with coefficient vectors

$$\mathbf{c}_i = (c_{0,i}, c_{1,i}, c_{2,i})^T.$$

Polynomials of degree n given in standard (i.e., non-homogeneous) form are homogenized by replacing t^i with $t_0^{n-i} t_1^i$.

[1] If improper parameterizations are given, one may obtain proper ones by applying a suitable reparameterization, see [9].

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

Furthermore we assume that both curves are in reduced form, i.e.

$$\gcd(p_0(\mathbf{t}), p_1(\mathbf{t}), p_2(\mathbf{t})) = \gcd(p_0'(\mathbf{t}), p_1'(\mathbf{t}), p_2'(\mathbf{t})) = 1$$

and of common degree $n \geq 2$. Affine transformation do not change the degree of a curve.

In particular, this implies that both curves possess the same degree

$$\max(\deg_{t_i}(p_0(\mathbf{t})), \deg_{t_i}(p_1(\mathbf{t})), \deg_{t_i}(p_2(\mathbf{t}))) = n,$$
$$\max(\deg_{t_i}(p_0'(\mathbf{t})), \deg_{t_i}(p_1'(\mathbf{t})), \deg_{t_i}(p_2'(\mathbf{t}))) = n,$$

with respect to t_i, $i = 0, 1$. Note that $n \geq 2$ exludes lines, since we consider proper parameterizations only.

Recall that using homogeneous coordinates allows to represent any affine transformation by a matrix multiplication

$$\mathbf{x} \mapsto M\mathbf{x}, \quad M = \begin{pmatrix} 1 & 0 \\ \vec{\mathbf{b}} & A \end{pmatrix},$$

where A is a 2×2 matrix and $\vec{\mathbf{b}} \in \mathbb{R}^2$. Any regular affine transformation is represented by a non-singular matrix M. The class of affine transformations includes translations, rotations, uniform and non-uniform scalings, reflections and shears.

Definition 1 Two curves $\mathcal{C}$ and $\mathcal{C}'$ are said to be *affinely equivalent* if there exists a regular affine transformation matrix M such that $\mathcal{C}' = M\mathcal{C}$. Furthermore, $\mathcal{C}$ is said to possess an *affine symmetry* if there exists a regular affine transformation matrix M, different from the identity, such that $\mathcal{C} = M\mathcal{C}$.

Due to the group structure of regular affine mappings, affine equivalences define an equivalence relation. If the matrix A is orthogonal, i.e. $A^TA = I$, then affinely equivalent curves are said to be *congruent* and an affine symmetry is simply called a *symmetry*. If A is a multiple of an orthogonal matrix, $A^TA = \lambda I$ with $\lambda \in \mathbb{R}$, then the affinely equivalent curves are said to be *similar*.

2.2 Coefficient-based detection

Lemma 1 *Two rational parameterizations* $\mathbf{p}(\mathbf{t})$ *and* $\mathbf{p}'(\mathbf{t})$ *are equivalent, i.e.* $\mathbf{p}(\mathbf{t}) \simeq \mathbf{p}'(\mathbf{t})$ *holds for all* $\mathbf{t} \in P^1(\mathbb{R})$*, if and only if there exists a non-zero constant* μ *such that* $\mathbf{c}_i = \mu\mathbf{c}_i'$*,* $i = 0, \ldots, n$*.*

Proof. The equivalence of the two curves implies that there exists a rational function

$$\mu(\mathbf{t}) = \frac{\mu_1(\mathbf{t})}{\mu_0(\mathbf{t})} = \frac{p_0'(\mathbf{t})}{p_0(\mathbf{t})} = \frac{p_1'(\mathbf{t})}{p_1(\mathbf{t})} = \frac{p_2'(\mathbf{t})}{p_2(\mathbf{t})}$$

where μ_0 are μ_1 are relatively prime polynomials, such that $\mathbf{p}(\mathbf{t}) = \mu(\mathbf{t})\mathbf{p}'(\mathbf{t})$. Consequently, the two rational curves satisfy

$$\mu_0(\mathbf{t})\mathbf{p}(\mathbf{t}) = \mu_1(\mathbf{t})\mathbf{p}'(\mathbf{t}).$$

This function is indeed a constant since

$$\mu_0 | \underbrace{\gcd(p_0', p_1', p_2')}_{=1} \quad \text{and} \quad \mu_1 | \underbrace{\gcd(p_0, p_1, p_2)}_{=1}. \qquad \square$$

Recall that any two proper parameterizations of a rational curve are related by a linear rational reparameterization, which is simply a regular projective transformation of the real projective line

$$\mathbf{r}(\mathbf{t}) = \underbrace{\begin{pmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{pmatrix}}_{=\alpha} \mathbf{t} = \begin{pmatrix} \alpha_{00}t_0 + \alpha_{01}t_1 \\ \alpha_{10}t_0 + \alpha_{11}t_1 \end{pmatrix}$$

described by a regular matrix α. We investigate the transformation of the coefficients which is caused by such a reparameterization.

Lemma 2 *The reparameterized curve* $\hat{\mathbf{p}} = \mathbf{p} \circ \mathbf{r}$*,*

$$\mathbf{p}(\mathbf{r}(\mathbf{t})) = \hat{\mathbf{p}}(\mathbf{t}) = \sum_{j=0}^{n} \hat{\mathbf{c}}_j t_0^{n-j} t_1^j$$

has the coefficients

$$\hat{\mathbf{c}}_j(\alpha) = \sum_{i=0}^{n} \mathbf{c}_i \sum_{\ell=0}^{j} \binom{n-i}{\ell} \binom{i}{j-\ell} \alpha_{00}^{n-i-\ell} \alpha_{01}^{\ell} \alpha_{10}^{i-j+\ell} \alpha_{11}^{j-\ell}$$

for $j = 0, \ldots, n$*.*

Proof. This result is confirmed by a simple computation and by comparing the coefficients. $\square$

We identify affine equivalences by analyzing whether the coefficients are related by an affine transformation.

Proposition 3 *Let* $\mathcal{C}$ *and* $\mathcal{C}'$ *be rational planar curves with parameterizations* $\mathbf{p}(t)$ *and* $\mathbf{p}'(t)$ *satisfying our assumptions. The two curves are affinely equivalent if and only if there exists a constant* μ*, an affine transformation matrix* M *and a regular projective transformation* α*, such that the control points of both curves satisfy*

$$M\mathbf{c}_j' = \mu\, \hat{\mathbf{c}}_j(\alpha), \quad j = 0, \ldots, n. \tag{2}$$

Proof. On the one hand, the conditions (2) imply that the two curves are affinely equivalent. On the other hand, we consider two affinely invariant curves $\mathcal{C}'$ and $\mathcal{C}$. There exists an affine transformation M such that

$$M\mathcal{C}' = \mathcal{C}.$$

We define $\mathbf{z}(\mathbf{t}) = M\mathbf{p}'(\mathbf{t})$. Consequently $\mathbf{z}(\mathbf{t})$ and $\mathbf{p}(\mathbf{t})$ are two proper parameterizations of the same curve $\mathcal{C}$. According to Lemma 4.17 of [9] there is a linear rational reparameterization $\mathbf{r}(\mathbf{t})$ – and hence an associated projective transformation α – such that

$$\mathbf{z}(\mathbf{t}) \simeq \mathbf{p}(\mathbf{r}(\mathbf{t})).$$

Thus we obtain that

$$\sum_{i=0}^{n} M\mathbf{c}'_i t_0^{n-i} t_1^i = M\mathbf{p}'(\mathbf{t}) = \mathbf{z}(\mathbf{t}) \simeq \mathbf{p}(\mathbf{r}(\mathbf{t})) = \hat{\mathbf{p}}(\mathbf{t}) = \sum_{i=0}^{n} \hat{\mathbf{c}}_i(\alpha) t_0^{n-i} t_1^i.$$

Using Lemma 1 confirms (2). □

2.3 Barycentric coordinates

The existence of the affine transformation matrix M can be characterized with the help of barycentric coordinates.

Corollary 4 *Let $\mathcal{C}$ and $\mathcal{C}'$ be two rational planar curves as in Proposition 3. We assume that*

(i) all points $\mathbf{c}'_i$ are finite points ($c'_{i,0} \neq 0$) and

(ii) the first three points $\mathbf{c}'_0$, $\mathbf{c}'_1$ and $\mathbf{c}'_2$ are non-collinear.

The two curves $\mathcal{C}$ and $\mathcal{C}'$ are affinely equivalent if and only if there exist a regular projective transformation α and a constant μ such that the equations

$$c'_{j,0} = \mu\, \hat{c}_{j,0}(\alpha), \quad j = 0, \ldots, n \tag{3}$$

and

$$\lambda_i(\mathbf{c}'_j; \mathbf{c}'_0, \mathbf{c}'_1, \mathbf{c}'_2) = \lambda_i(\hat{\mathbf{c}}_j(\alpha); \hat{\mathbf{c}}_0(\alpha), \hat{\mathbf{c}}_1(\alpha), \hat{\mathbf{c}}_2(\alpha)), \quad i = 0, \ldots, 2, \quad j = 3, \ldots, n. \tag{4}$$

are satisfied.

For any solution of the system (3) and (4), we obtain the corresponding affine transformation by solving the linear system of equations in six unknowns

$$M\mathbf{c}'_i = \mu\hat{\mathbf{c}}_i \qquad \text{for } i = 0, \ldots, 2.$$

In order to find Euclidean congruences and Euclidean symmetries (resp. similarities) we have to check in a postprocessing step whether the submatrix A is orthogonal (resp. a multiple of an orthogonal matrix).

Clearly, it is also possible to consider other triplets of points in (ii) and (4).

2.4 The case of Bézier control points

Rational Bézier curves of degree n

$$\mathbf{p}(u) = \sum_{i=0}^{n} B_i^n(u)\mathbf{b}_i$$

generally possess the properties (proper parameterization, reduced form, common denominator) which are assumed by our method. These curves can be homogenized by simply replacing the Bernstein polynomials $B_i^n(u)$ by $\binom{n}{i} t_0^{n-i} t_1^i$. This is equivalent to the standard homogenization $u = \frac{u_1}{u_0}$ and a multiplication by u_0^n, followed by the projective transformation

$$\begin{pmatrix} t_0 \\ t_1 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} u_0 - u_1 \\ u_1 \end{pmatrix}.$$

of the parameter domain. That means that the control points $\mathbf{b}_i$ of the Bézier curves are related to the monomial coefficients after this transformation via

$$\mathbf{c}_i = \binom{n}{i} \mathbf{b}_i.$$

3 Implementation and Examples

If the two conditions (i) or (ii) are not all satisfied then we apply an arbitrary projective transformation α' to the parameterization $\mathbf{p}'$ of the second curve. Note that the first condition is always violated for polynomial curves, hence a reparameterization is needed in this situation. The reparameterized curve $\hat{\mathbf{p}}' = \mathbf{p}' \circ \mathbf{r}'$, where $\mathbf{r}'$ is defined by α', satisfies all conditions in general and its control points are obtained from Lemma 2.

From equations (3) and (4) we obtain a system of polynomial equations in α and μ by using (1). Without loss of generality we may apply the normalization $|\mu| = 1$ and arrive at the equations

$$c'_{j,0} = \pm\, \hat{c}_{j,0}(\alpha), \quad j = 0, \ldots, n$$

and

$$\lambda_i(\mathbf{c}'_j; \mathbf{c}'_0, \mathbf{c}'_1, \mathbf{c}'_2) = \lambda_i(\hat{\mathbf{c}}_j(\alpha); \hat{\mathbf{c}}_0(\alpha), \hat{\mathbf{c}}_1(\alpha), \hat{\mathbf{c}}_2(\alpha)), \quad i = 0, \ldots, 2, \quad j = 3, \ldots, n.$$

These form a system in four unknowns α consisting of $3n - 3$ equations, since we may omit the equations obtained for $i = 2$ as the barycentric coordinates sum to 1.

We performed the computations using Mathematica Version 10, where we used the built-in functions Reduce[] and NSolve[] to solve the system by symbolic and numeric computations, respectively. For every example we considered affine symmetries, and affine equivalences with ($\mathbf{p}'$) and without ($\mathbf{p}''$) reparameterization. More precisely, we considered a master curve and two curves derived from it by applying affine transformations and parameter transformations.

The first example is the lemniscate (Fig. 1), which is a degree 4 curve given by

$$t \mapsto \begin{pmatrix} 1 + 4t + 12t^2 + 16t^3 + 8t^4 \\ 1 + 4t + 6t^2 + 4t^3 \\ 2t + 6t^2 + 4t^3 \end{pmatrix}.$$

example	deg.	# of equiv.	$\mathbf{p}(\mathbf{t})$ NSolve	$\mathbf{p}(\mathbf{t})$ Reduce	$\mathbf{p}(\mathbf{t})$ and $\mathbf{p}'(\mathbf{t})$ NSolve	$\mathbf{p}(\mathbf{t})$ and $\mathbf{p}'(\mathbf{t})$ Reduce	$\mathbf{p}(\mathbf{t})$ and $\mathbf{p}''(\mathbf{t})$ NSolve	$\mathbf{p}(\mathbf{t})$ and $\mathbf{p}''(\mathbf{t})$ Reduce
lemniscate	4	4	0.33	0.25	0.33	0.23	0.41	0.45
epitrochoid	4	2	0.14	0.3	0.13	0.3	0.14	0.34
4-leaf rose	6	8	2.65	2.53	2.68	1.97	2.34	2.45
offset of a cardioid	8	2	1.03	20.04	1.03	19.91	1.06	31.76

Table 1: Computation times (times in seconds)

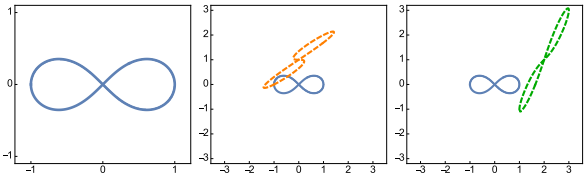


Figure 1: Several lemniscate like curves

As a second example we investigate the epitrochoid (see Fig. 2), given by

$$t \mapsto \begin{pmatrix} 7 + 28t + 56t^2 + 56t^3 + 28t^4 \\ 1 + 4t + 24t^2 + 40t^3 + 12t^4 \\ 4t + 12t^2 - 8t^3 - 16t^4 \end{pmatrix}.$$

Again we applied reparameterizations and affine mappings (not shown), similar to the previous example.

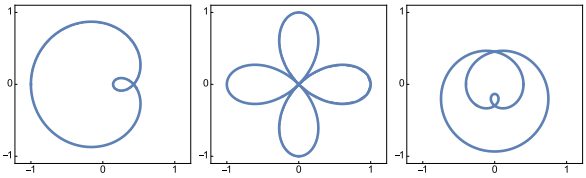


Figure 2: Epitrochoid, 4-leaf rose and offset of a cardioid.

The third example, the 4-leaf rose (see again Fig. 2), is given by the parameterization

$$t \mapsto \begin{pmatrix} 11 + 6t + 18t^2 + 32t^3 + 36t^4 + 24t^5 + 8t^6 \\ 2t + 10t^2 + 8t^3 - 16t^4 - 24t^5 - 8t^6 \\ 1 + 6t + 8t^2 - 8t^3 - 20t^4 - 8t^5 \end{pmatrix}$$

of degree 6. Finally we apply our algorithm to the offset of a cardioid (see again Fig. 2),

$$t \mapsto$$
$$\begin{pmatrix} 15(6561 + 2916t^2 + 486t^4 + 36t^6 + t^8) \\ -39366 + 61236t^2 - 31104t^3 + 3456t^5 - 756t^6 + 6t^8 \\ -18t(4374 - 1296t - 1134t^2 + 864t^3 - 126t^4 - 16t^5 + 6t^6) \end{pmatrix}$$

which is a rational curve of degree 8.

Table 1 presents the computation times (on a standard PC) for solving the system (3) and (4) in these examples.

4 Conclusion

We presented a method to detect affine equivalences of planar rational curves. For moderate degrees of the input curve, the corresponding polynomial system can be solved within seconds using standard computer algebra tools. To the best of our knowledge, this is the first work on detecting affine equivalences and affine symmetries of rational curves, and it also encompasses the computation of symmetries or similarities, which was studied by several authors [1, 2, 3, 4, 5, 7, 8], as special cases. Future work will be devoted to the generalization to higher dimensions and to the detection of approximate affine equivalences via numerical methods.

References

[1] Juan Gerardo Alcázar. Efficient detection of symmetries of polynomially parametrized curves. *J. Comput. Appl. Math.*, 255:715–724, 2014.

[2] Juan Gerardo Alcázar, Carlos Hermoso, and Georg Muntingh. Detecting similarity of rational plane curves. *J. Comput. Appl. Math.*, 269:1–13, 2014.

[3] Juan Gerardo Alcázar, Carlos Hermoso, and Georg Muntingh. Detecting symmetries of rational plane and space curves. *Comput. Aided Geom. Des.*, 31(3-4):199–209, 2014.

[4] Juan Gerardo Alcázar, Carlos Hermoso, and Georg Muntingh. Symmetry detection of rational space curves from their curvature and torsion. *Comput. Aided Geom. Des.*, 33(0):51 – 65, 2015.

[5] Peter Braß and Christian Knauer. Testing congruence and symmetry for general 3-dimensional objects. *Comput. Geom.*, 27(1):3–11, 2004.

[6] Z. Huang and F. S. Cohen. Affine-invariant B-spline moments for curve matching. *Trans. Img. Proc.*, 5(10):1473–1480, October 1996.

[7] Peter Lebmeir and Jürgen Richter-Gebert. Rotations, translations and symmetry detection for complexified curves. *Comput. Aided Geom. Des.*, 25(9):707–719, 2008.

[8] J. Sánchez-Reyes. Detecting symmetries in polynomial Bézier curves. *J. Comput. Appl. Math.*, 288:274–283, 2015.

[9] J. Rafael Sendra, Franz Winkler, and Sonia Pérez-Díaz. *Rational algebraic curves. A computer algebra approach.* Berlin: Springer, 2008.

Robustness of Zero Sets: Implementation

Peter Franek[*] Marek Krčál[†] Hubert Wagner[‡]

Abstract

Robustness of zero of a continuous map $f\colon X \to \mathbb{R}^n$ is the maximal $r > 0$ such that each $g\colon X \to \mathbb{R}^n$ with $\|f - g\|_\infty \leq r$ has a zero. We develop and implement an efficient algorithm approximating the robustness of zero and present computational experiments.

The main ingredient is an algorithm for deciding the topological extension problem based on computing cohomological *obstructions* to extendability and their robustness.

1 Introduction

Statement of the result. We describe an algorithm for detecting zeros of vector valued functions $f : X \to \mathbb{R}^n$ on a compact space X and for approximating the *robustness* of zero, that is, a maximal number $r > 0$ such that every continuous $g : X \to \mathbb{R}^n$ satisfying $\|g - f\| \leq r$ has a zero. By $\|f\|$ we denote the max norm of f, that is, $\max_{x\in X} |f(x)|$ where $|\cdot|$ is a fixed ℓ_p norm in $\mathbb{R}^n$. Nontrivial cases happen if $\dim X \geq n$, as otherwise arbitrarily small perturbations of f avoid zero.

For computer representation we assume that the space X is a simplicial complex. Then the map $f\colon X \to \mathbb{R}^n$ is specified by its values on the vertices and by a value $\alpha > 0$ such that $|f(x) - f(y)| \leq \alpha$ for arbitrary points x and y of any simplex of X. In an alternative setting we might assume that the function f is *simplexwise linear*,[1] but we preferred to emphasize that the precise knowledge of f is not needed (at the cost of slightly worse approximation guarantees).

The main motivation for the theoretical part of this paper was to give a rigorous analysis of an implementation that is tailored for real instances. We perceive the contribution of this paper as follows.

- *Feasibility.* Our algorithm is designed to avoid any time-costly numerical computations. Unlike the algorithm of [3], we need neither barycentric subdivisions nor convex optimization.
- *Persistent cohomology computations over integers.* As an auxiliary tool, we need to extract certain information from a persistent module $H^*(X_0;\mathbb{Z}) \to H^*(X_1;\mathbb{Z}) \to \dots$ with integral coefficients. To that end, we adapted the Chen's and Kerber's matrix reduction algorithm "with a twist" [2].
- *Implementation.* Our implementation is available online[2] and several computational experiments are presented in our preprint [4].

Methods and the outline of the algorithm. The main tools come from the field of *computational homotopy theory*. In [3] we showed that any function $f\colon X \to \mathbb{R}^n$ on a compact domain X, has an r-robust zero if and only if the map

$$\begin{aligned} f|_{A(r)} :& A(r) \to \mathbb{R}^n \setminus \{0\} \text{ where} \\ & A(r) := \{x \in X : \ |f(x)| \geq r\} \end{aligned} \tag{1}$$

cannot be extended to a map $X \to \mathbb{R}^n \setminus \{0\}$. After replacing f by the map $x \mapsto f(x)/|f(x)|$, we can equivalently replace $\mathbb{R}^n \setminus \{0\}$ by S^{n-1}. This extension problem is the core of our algorithm for approximating robustness, outlined as follows.

A. First, we discretize the continuous input, that is, convert the spaces $A(r)$ into simplicial complexes A_r. Unlike in [3], we do not aim at having homotopy equivalence $A(r) \simeq A_r$ which requires additional subdivisions and thus increases the computing time heavily. For obtaining approximate results it is sufficient to have a relation of the form $A(r - \alpha) \supseteq A_r \supseteq A(r + \alpha)$ for some reasonably small α. Such a relation can be achieved without introducing additional subdivisions while using the simplexwise Lipschitz property of f.

   We also identify some smallest value r_0 such that the restriction of f to A_{r_0} can be easily *discretized*. A simple combinatorial procedure finds a simplicial map f' from A_{r_0} to the sphere such that f' is homotopic to f as map from A_{r_0} to the $(n-1)$-sphere.

B. In the second step, we do pure computational homotopy theory. Namely, for a previously obtained sequence of simplicial complexes $X \supseteq$

[*]IST Austria. Email: peter.franek@gmail.com This research has been supported by Austrian Science Fund (FWF): M 1980

[†]IST Austria. Email: marek.krcal@ist.ac.at

[‡]IST Austria. Email: hub.wag@gmail.com

[1] That is, on every simplex it linearly interpolates the values on the vertices. Such functions defined on sufficiently fine subdivision of X can approximate any continuous map $X \to \mathbb{R}^n$ arbitrarily well.

[2] www.cs.cas.cz/~franek/rob-sat

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

$A_0 \supseteq A_1 \supseteq \ldots \supseteq A_h = \emptyset$ and a simplicial sphere-valued map f' we ask for robustness of non-extendability of f' defined as follows.

Definition 1 *Let $X \supseteq A_0 \supseteq A_1 \supseteq \ldots$ be a filtration and $f' : A_0 \to S^{n-1}$ a sphere-valued map that cannot be extended to all of X. The* robustness of non-extendability of f' from $(A_i)_{i\geq 0}$ to X *is the smallest index i such that $f'|_{A_i}$ cannot be extended to the whole of X.*

2 Discretizing the geometry of the zero sets of perturbations

Definition 2 *A* continuous filtration *of spaces is a family $(A_r)_{r\in\mathbb{R}}$ such that $A_r \supseteq A_s$ whenever $r \leq s$.*

A continuous filtration $(A_r)_{r\in\mathbb{R}}$ is called step-like *whenever there exists a sequence of numbers $-\infty =: r_{-1} < r_0 \leq r_1 \leq r_2 \leq \ldots \leq r_k$ such that for any $r, s \in (r_i, r_{i+1}]$ holds $A_r = A_s$ for all i.*

Note that any such step-like continuous filtration is determined by the sequence of reals $(r_i)_i$ and the filtration $A_0 \supseteq A_1 \supseteq \ldots \supseteq A_k$ where each A_i denotes A_{r_i}.

Definition 3 *Continuous filtrations $(A_r)_r$ and $(B_r)_r$ are called α-interleaved whenever $B_{r+\alpha} \subseteq A_r$ and $A_{r+\alpha} \subseteq B_r$ for each $r \in \mathbb{R}$.*

Definition 4 *Let $f\colon X \to \mathbb{R}^n$ be a continuous map on a simplicial complex X and let $|\cdot|$ be a norm on $\mathbb{R}^n$.*

1. *Then by A_r we denote the subcomplex of X spanned by the vertices v of X with $|f(v)| \geq r$.*
2. *By $A(r)$ we denote the subspace of X defined by $A(r) = \{x \in X\colon |f(x)| \geq r\}$.*
3. *We say that f is* simplexwise α-Lipschitz *whenever $|f(x) - f(y)| \leq \alpha$ for each pair of points $x, y \in \Delta$ of any simplex $\Delta \in X$.*

Spaces A_r form a step-like filtration where the steps occur for each r equal to $|f(v)|$ for some vertex v of X.

We will represent the sphere S^{n-1} via a simplicial complex Σ^{n-1}, the boundary of the n-dimensional cros-polytope. Denoting $e_1, \ldots, e_n$ the canonical basis vectors of $\mathbb{R}^n$, simplices of Σ^{n-1} are all those subsets of $\{\pm e_i \,|\, i = 1, \ldots, n\}$ that do not contain a pair of antipodal vectors $\{e_i, -e_i\}$.

Theorem 1 *Let $f\colon X \to \mathbb{R}^n$ be a simplexwise α-Lipschitz map for some constant $\alpha > 0$. Then the following holds:*

1. *The continuous filtrations $(A_r)_{r\in\mathbb{R}}$ and $A(r)_{r\in\mathbb{R}}$ are α-interleaved.*
2. *For any ℓ_p norm once $r > \alpha n^{1/p}/2$, the mapping of vertices*

$$\begin{aligned} f' : V(A_r) &\to V(\Sigma^{n-1}) \\ v &\mapsto \operatorname{sgn}\big(f(v)_{i^*}\big) e_{i^*} \\ \text{where } \ i^* &= \operatorname*{argmax}_{i=1,\ldots,n} \big|f(v)_i\big| \end{aligned} \tag{2}$$

defines a simplicial map $f'\colon A_r \to \Sigma^{n-1}$ (that is, it maps simplices to simplices).

Moreover, $f'\colon A_r \to \Sigma^{n-1} \subseteq \mathbb{R}^n \setminus \{0\}$ is homotopic to $f|_{A_r}\colon A \to \mathbb{R}^n \setminus \{0\}$ once $r > \alpha n^{1/p}$.

The simplicial map $f'\colon A \to \Sigma^{n-1}$ as above will be called the *simplicial approximation of $f|_A$*.

3 The algorithm using an oracle for robustness of non-extendability.

Now it is convenient to state the algorithm for approximating robustness of zero, given an oracle for computing or bounding from below the robustness of non-extendability.

A. (a) Label the set of real values $\{|f(v)|\colon v \in V(X)$ such that $|f(v)| \geq \alpha n^{1/p}\}$ by $\{r_0, r_1, \ldots, r_h\}$ for some integer $h \geq 0$.

(b) For any simplex $\Delta \in X$ compute its filtration value $r(\Delta)$ by

$$r(\Delta) := \min_{v \text{ vertex of } \Delta} |f(v)|.$$

This yields a filtration $A_0 = A_{r_0} \supseteq \ldots \supseteq A_h = A_{r_h}$ that together with the values $r_0, \ldots, r_h$ determines the step-like continuous filtration $(A_r)_{r\in\mathbb{R}}$ from Definition 4.

(c) For vertices v of X with $|f(v)| \geq r_0$ compute $f'(v)$ defined by (2).

B. Use oracle to compute or bound from below the robustness i^* of non-extendability of f' from $(A_i)_{i\geq 0}$ to X.

(a) Once $i^* \geq j$ and $j > 0$, output "robustness of zero is at least $r_j - \alpha$."

(b) Once also $i^* \leq j$, output "robustness of zero is at most $r_j + \alpha$."

Using the fact that the nonextendability of (1) is equivalent to the existence of an r-robust zero combined with Theorem 1, we can easily prove that the above algorithm outputs a correct statement.

4 Robustness of obstructions to extendability.

Here we review the basic facts from obstruction theory. $X^{(k)}$ will always refer to the k-skeleton of X, the sub-complex spanned by simplices of dimension

$\leq k$. Any map $f : A \to S^{n-1}$ can be extended to $A \cup X^{(n-1)} \to S^{n-1}$ by the connectivity of the sphere. At some point of our extension process we need to work on the level of cocycles. Also our implementation operates fully on the level of cochains and cocycles, hence we stick to that point of view for most of the exposition as well. Let $z \in Z^{n-1}(\Sigma^{n-1}, \mathbb{Z})$ be a fixed representative of the generator of $H^{n-1}(\Sigma^{n-1}; \mathbb{Z})$. We will use the following facts:

Proposition 2 *Let $f\colon A \to \Sigma^{n-1}$ be simplicial, $X \supseteq A$, and $y := f^\sharp(z) \in Z^{n-1}(A; \mathbb{Z})$. Then the following holds:*

1. *Any map $h\colon A \cup X^{(n-1)} \to \Sigma^{n-1}$ extendable to $X^{(n)}$ such that $h|_A = f$ can be described up to a homotopy stationary on A by a cocycle $x \in Z^{n-1}(X; \mathbb{Z})$ such that $x|_A = y$. If $n \leq 2$, then any map $A \cup X^{(n)} \to \Sigma^{n-1}$ extends to all of X.*

2. *If $n \geq 3$, for any $x \in Z^{n-1}(X; \mathbb{Z})$ such that $x|_A = y$ we have that $x \smile_{n-3} x$ vanishes[3] on A, that is, it is element of $Z^{n+1}(X, A; \mathbb{Z}_2)$ (or element of $Z^4(X, A; \mathbb{Z})$ for $n = 3$) and it is a relative coboundary if and only if the corresponding map h can be extended to a map $X^{(n+1)} \to \Sigma^{n-1}$.*

We will use the notation $\Omega := \{x \in Z^{n-1}(X; \mathbb{Z})\colon x|_A = y\}$ further below. Test 1. above (corresponding to the primary obstruction) directly translates to an algorithm and the second one (the secondary obstruction) does so as well once $n > 3$. (We also explain what the notions of the primary and secondary obstructions mean exactly below.)

1. The set Ω corresponds to solutions of a linear equation over integers. To see that, let $\bar{y} \in C^{n-1}(X; \mathbb{Z})$ be an arbitrary cochain such that $\bar{y}|_A = y$. We have that

$$\Omega = \{\bar{y} - c\colon c \in C^{n-1}(X, A; \mathbb{Z}) \text{ such that } \delta c = \delta \bar{y}\}. \tag{3}$$

   Thus the extendability of f to $X^{(n)} \to \Sigma^{n-1}$ is equivalent to solvability of the linear equation $\delta c = \delta \bar{y}$ with the unknown $c \in C^{n-1}(X, A; \mathbb{Z})$.

2. Once Ω is nonempty, we fix $x \in \Omega$. From (3) it follows that there is a bijection $Z^{n-1}(X, A; \mathbb{Z}) \to \Omega$ given by $w \mapsto x - w$. Thus the extendability of f to a map $X^{(n+1)} \to \Sigma^{n-1}$ is equivalent to the existence of $w \in Z^{n-1}(X, A; \mathbb{Z})$ such that

$$\begin{aligned}&[(x - w) \smile_{n-3} (x - w)] = \\ &= [x \smile_{n-3} x] - [w \smile_{n-3} w] = \\ &= 0 \in H^{n+1}(X, A; \mathbb{Z}_2).\end{aligned}$$

   We use that the map $w \mapsto w \smile_{n-3} w$ induces a homomorphism on the level of cohomology for $n > 3$, therefore the question reduces to a system of linear equations again. This formulation shows that the coset

$$[x \smile_{n-3} x] + \underbrace{Sq^2\big(H^{n-1}(X, A; \mathbb{Z})\big)}_{\{[w \smile_{n-3} w]\colon w \in Z^{n-1}(X, A; \mathbb{Z})\}}$$

   of $H^{n+1}(X, A; \mathbb{Z}_2)$—called the *secondary obstruction*—captures the lack of extendability to the $(n+1)$st skeleton $X^{(n+1)}$. In the case $n = 3$ the extendability condition is $[(x - w) \smile (x - w)] = 0 \in H^4(X, A; \mathbb{Z})$ for some w which is computationally equivalent to solving systems of quadratic Diophantine equations—an undecidable problem [5]. In many instances, the quadratic equations are simple if not trivial and very simple heuristics suffice to solve them.

Robustness of the primary and secondary obstruction. In the persistent setting the input contains, in addition to above, a sequence of spaces $A_0 = A, A_1, \ldots, A_h$ and we want to compute a lower-bound on the robustness of non-extendability—a value k such that $f|_{A_k}$ cannot be extended to X for as large k as possible.

The key concept that allows an easy modification of the obstruction tests into the persistent setting is the functoriality of cohomology. For instance, the cochain extension $\bar{y}$ of $f^\sharp(z)$ is an extension of $f^\sharp|_{A_i}(z)$ for each $A_i \subseteq A$. The same holds for the cocycle extension x.

We state the algorithm **Primary–Secondary Persistence** for lower-bounding the robustness of non-extendability on a high-level fashion that emphasizes *what the algorithm does* rather that *how it is done*. The low level implementation is explained in the preprint [4].

0. Compute $y := f^\sharp(z) \in Z^{n-1}(A_0; \mathbb{Z})$. Fix an arbitrary extension $\bar{y} \in C^{n-1}(X; \mathbb{Z})$ of y.

1. Find the smallest $j \geq 0$ such that there is $c \in C^{n-1}(X, A_j; \mathbb{Z})$ such that $\delta c = \delta \bar{y}$. If $n = 1, 2$ output j. Otherwise let $x := \bar{y} - c$.

2. Find the smallest $k \geq j$ such that there is $b \in C^n(X, A_k; \mathbb{Z}_2)$ and $w \in Z^{n-1}(X, A_k; \mathbb{Z})$ such that $\delta b + w \smile_{n-3} w = x \smile_{n-3} x$. Output k.

Step 0 amounts to using the definition of the induced map in simplicial cohomology: namely, $f^\sharp(z)$ is defined to evaluate to 1 on the simplices $[v_1, \ldots, v_n]$ such that $[f(v_1), \ldots, f(v_n)] = [e_1, \ldots, e_n]$ and to evaluate to 0 once $\{f(v_1), \ldots, f(v_n)\} \neq \{e_1, \ldots, e_n\}$.

Step 1 and 2 are more involved and are reduced to matrix reductions over integers similar to those used in persistent homology computations over fields.

[3] By $x \smile_{n-3} x$ we denote a cocycle representant of the Steenrod square $Sq^2[x]$.

5 Experimental results with random fields.

One of our goals is to analyze how much "typical" is a situation in which the secondary obstruction or higher obstructions play a role. The lowest-dimensional case where nontrivial secondary obstruction can occur is $(m, n) = (4, 3)$. We generated random continuous functions $f : [-1, 1]^4 \to \mathbb{R}^3$ taken from different probability distributions and looked for possible nontrivial secondary obstructions. However, while the primary obstruction typically occurs whenever f contains a zero, we couldn't detect a single instance of a randomly generated function with nontrivial higher obstruction. Still, we don't dare to conclude that higher obstructions are untypical or unnatural, and think that more research is needed.[4] A short description of our first experiments follows.

First we considered random functions generated as Gaussian random fields. Each component $f_i(x)$ of $f(x)$ was generated so that for any finite set of points $\{x_1, \ldots, x_k\}$ the random vector $\{f_i(x_j) : j = 1, \ldots, k\}$ has a multivariate normal distribution with mean zero and the covariance between $f_i(x)$ and $f_i(y)$ was taken to be

$$C(x, y) = \exp(-\frac{|x - y|^2}{2l^2})$$

for suitable $l > 0$. We generated function values using $l = 1/2$, sampled from a grid $g^4 = 28^4 \subseteq [-1, 1]^4$ with the three components of f generated independently.

For each trial, we first computed the minimal r_0 for which $f'|_{A^\square_{r_0}}$ is simplicial.[5] From a sample of 1218 functions, the average value of the minimal simplicial r_0 was 0.46. This value could be made smaller by refining the grid: however, in all cases, there was a nontrivial primary obstruction which persisted up to $r_1 > r_0$ whose value was in average 1.06. In all but three cases, there was no potential for a nontrivial secondary obstruction, because the cohomology group $H^4(X, A^\square_{r_1})$ was trivial. It was nontrivial in three cases, giving some hope for a nontrivial secondary obstruction, but there was no secondary obstruction in these cases either.[6]

One possible explanation for the lack of secondary obstruction is that the cohomology in dimension 4 has typically lower robustness than in dimension 3 and most generators have already died when the primary obstruction (element of H^3) dies. A similar phenomenon occurs in persistent homology of excursion sets of random scalar fields, where the persistence barcodes in dimension 0 die before the barcodes in dimension 1, compare [1]. The lack of top dimensional cohomology reflects the fact that most components of the zero set intersect the boundary of the domain: it will be a matter of future work to perform similar computations for functions defined on manifolds without boundary.

We also tried to detect higher obstructions in the vector fields $f(x) - f(0)$ where f was generated as above and 0 is the midpoint of the $[-1, 1]^4$ cube, with the hope of isolating the zero set farther from the boundary. The top cohomology was indeed richer, but no secondary obstruction was detected either. We also tried to use other covariance functions but the results were similar.

Our last attempt to detect secondary obstruction in random fields was to generate random homogenous quadratic polynomials. The coefficients a^k_{ij} in $f_k(x) = \sum_{i,j} a^k_{i,j} x_i x_j$ were generated as independent samples from a standard normal distribution.[7] The zero set of homogenous quadratic functions is either the origin alone or a cone intersecting the boundary $\partial[-1, 1]^4$: only the first case can yield a nontrivial $H^4(X, A^\square_r)$ and a nontrivial secondary obstruction. We generated around 70 thousand instances of random quadratic functions on a 10^4 grid: around 2.2% of them had only the origin as the zero set, but there was no nontrivial secondary obstruction in a single instance.

[4] While we were not able to detect higher obstructions in random fields, they occur in relatively simple examples with component-wise quadratic functions.

[5] In our implementation, we work with a triangulation $A^\square_r$ of the cubical complex that consists of all cubes c such that $|f(x)| \geq r$ for all vertices of c, rather than with A_r defined above.

[6] We assume that in these three cases, nontriviality of $H^4(X, A^\square_r)$ was induced by a local positive minimum of $|f|$ in the interior of the domain, rather then by a neighborhood of zero set.

[7] This is motivated by the fact that simplest examples of functions with nontrivial secondary obstruction are quadratic and homogenous.

References

[1] Robert J. Adler, Omer Bobrowski, Matthew S. Borman, Eliran Subag, and Shmuel Weinberger. Persistent homology for random fields and complexes, 2010.

[2] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, 2011.

[3] P. Franek and M. Krčál. Robust satisfiability of systems of equations. *J. ACM*, 62(4):26:1–26:19, 2015.

[4] P. Franek, M. Krčál, and H. Wagner. Robustness of zero sets: Implementation, 2016. preprint `http://www.cs.cas.cz/~franek/rob-sat/experimental.pdf`.

[5] Yu. Matiyasevich. The Diophantineness of enumerable sets. *Dokl. Akad. Nauk SSSR*, 191:279–282, 1970.

Non-crossing Bottleneck Matchings of Points in Convex Position

Marko Savić* Miloš Stojaković*†

Abstract

Given an even number of points in a plane, we are interested in matching all the points by straight line segments so that the segments do not cross. Bottleneck matching is a matching that minimizes the length of the longest segment. For points in convex position, we present a quadratic-time algorithm for finding a bottleneck non-crossing matching, improving upon the best previously known algorithm of cubic time complexity.

1 Introduction

Let P be a set of n points in the plane, where n is an even number. Let M be a perfect matching of points in P, using $n/2$ straight line segments to match the points, that is, each point in P is an endpoint of exactly one line segment. We forbid line segments to cross. Denote the length of a longest line segment in M with $bn(M)$, which we also call the *value* of M. We aim to find a matching that minimizes $bn(M)$. Any such matching is called *bottleneck matching* of P.

1.1 Related work

There is plentiful research on various geometric problems involving pairings without crossings, see [4, 3, 5, 7, 6, 13]. The more basic of those problems involve matching pairs of points by straight line segments. It is a simple observation that there is always such a matching with non-crossing segments since it is straightforward to prove that a matching minimizing the total sum of lengths of its segments has to be non-crossing.

In [10], Chang, Tang and Lee gave an $O(n^2)$-time algorithm for computing a bottleneck matching of a point set, but in a context where crossings are allowed. This result was extended by Efrat and Katz in [12] to higher-dimensional Euclidean spaces.

Abu-Affash, Carmi, Katz and Trablesi showed in [2] that the problem of computing non-crossing bottleneck matching of a point set is NP-complete and does not allow a PTAS. They gave a $2\sqrt{10}$ factor approximation algorithm, and also showed that the case where all points are in convex position can be solved exactly in $O(n^3)$ time. In [1], Abu-Affash, Biniaz, Carmi, Maheshwari and Smid presented an algorithm for computing a non-crossing bottleneck plane matching of size at least $n/5$ in $O(n \log^2 n)$ time. They then extended it to provide an $O(n \log n)$-time approximation algorithm which computes a plane matching of size at least $2n/5$ whose edges have length at most $\sqrt{2} + \sqrt{3}$ times the length of a longest edge in a non-crossing bottleneck matching.

Bichromatic (sometimes also called bipartite) versions of the bottleneck matching problem, where only points of different colors are allowed to be matched, have also been studied. Efrat, Itai and Katz showed in [11] that a bottleneck matching between two point sets, with possible crossings, can be found in $O(n^{3/2} \log n)$ time. Bichromatic non-crossing bottleneck problem was proved to be NP-complete by Carlson, Armbruster, Bellam and Saladi in [9].

Biniaz, Maheshwari and Smid in [8] study special cases of non-crossing bichromatic bottleneck matchings. They show that the case where all points are in convex position can be solved in $O(n^3)$ time with an algorithm similar to the one for monochromatic case presented in [2]. They also consider the case where the points of one color lie on a line and all points of the other color are on the same side of that line, providing an $O(n^4)$ algorithm to solve it. The same results for these special cases are independently obtained in [9]. In [8] an even more restricted problem, a case where all points lie on a circle, is solved by constructing an $O(n \log n)$-time algorithm.

Here we only deal with matchings without crossings, so from now on, the word matching is used to refer only to pairings that are crossing-free.

1.2 Convex case and our result

In what follows we consider the case where all points of P are in convex position, i.e. they are the vertices of a convex polygon $\mathcal{P}$.

Let us label the points $v_0, v_1, \ldots, v_{n-1}$ in positive (counterclockwise) direction. To simplify the notation, we will often use only the indices when referring to the vertices. We write $\{i, \ldots, j\}$ to represent the sequence $i, i+1, i+2, \ldots, j-1, j$. All operations are calculated modulo n; note that i is not necessarily

*University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics. Partly supported by Ministry of Education and Science, Republic of Serbia. {marko.savic, milos.stojakovic}@dmi.uns.ac.rs

†Partly supported by Provincial Secretariat for Science, Province of Vojvodina.

less than j, and that $\{i, \ldots, j\}$ is not the same as $\{j, \ldots, i\}$. We say that (i, j) is a *feasible* pair if there exists a matching containing (i, j), which in this case simply means that $\{i, \ldots, j\}$ is of even size.

The problem of finding a bottleneck matching of points in convex position can be solved in polynomial time by a fairly straightforward dynamic programming algorithm, as presented in [2]. Similar algorithm for bichromatic case is presented in [8] and [9].

We present a faster algorithm for finding a bottleneck matching in the monochromatic case, with only $O(n^2)$ time complexity.

2 Structure of bottleneck matching

Our aim is to show the existence of a bottleneck matching with a certain structure that we can utilize to construct an efficient algorithm. We do so by proving a sequence of lemmas, with each lemma imposing an increasingly stronger condition on the structure.

Let us split all point pairs into the two categories. Pairs consisting of two neighboring vertices of $\mathcal{P}$ are called *edges*, and all other pairs are called *diagonals*. Each matching is, thus, comprised of edges and diagonals.

The *turning angle* of $\{i, \ldots, j\}$, denoted by $\tau(i, j)$, is the angle by which the vector $\overrightarrow{v_i v_{i+1}}$ should be rotated in positive direction to align with vector $\overrightarrow{v_{j-1} v_j}$, see Figure 1.

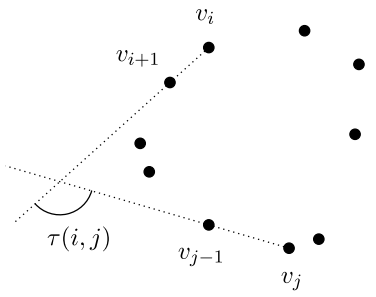


Figure 1: Turning angle.

We start by showing that there are bottleneck matchings satisfying the following constraint on turning angles.

Lemma 1 *There is a bottleneck matching M of P such that all diagonals $(i, j) \in M$ have $\tau(i, j) > \pi/2$.*

Let us consider the division of the polygon $\mathcal{P}$ into regions obtained by cutting it with diagonals (but not edges) of the given matching M. Each region in this division is bounded by some diagonals of M and by some edges from the polygon's boundary. If there are exactly k diagonals bounding a region, we say the region is *k-bounded*. Any maximal sequence of diagonals

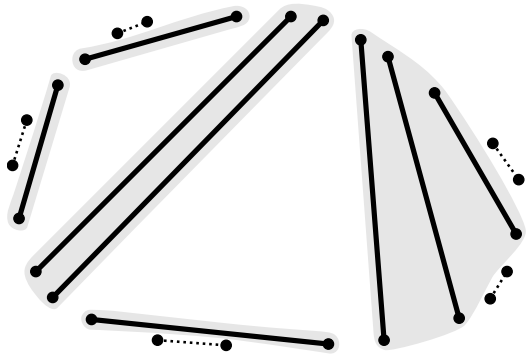


Figure 2: Diagonals inside each shaded area make a single cascade. There are three cascades with only one diagonal, one cascade with two diagonals, and one cascade with three diagonals.

connected by 2-bounded regions is called a *cascade*, see Figure 2. We can prove the following lemma.

Lemma 2 *There is a bottleneck matching having at most three cascades.*

From Lemma 2 we know that there is a bottleneck matching either without 3-bounded regions and at most one cascade, or with a single 3-bounded region and exactly three cascades. Obviously, it is not possible for a matching to have exactly two cascades. Next, we define a set of simpler problems that will be used to find an optimal solution in both of these cases.

3 Subproblems

Let Matching(i, j) be the problem of finding an optimal matching $M_{i,j}$ of points $\{i, \ldots, j\}$ only, so that $M_{i,j}$ has at most one cascade, and pair (i, j) belongs to a region bounded by at most one diagonal from $M_{i,j}$ different from (i, j).

If $j - i = 1$, then the solution to Matching(i, j) is exactly the edge (i, j). If $j - i > 2$, we consider the following cases. If there is a solution to Matching(i, j) that contains the pair (i, j), then $M_{i,j}$ can be constructed by taking (i, j) and $M_{i+1,j-1}$ together. If not, then at least one of the edges $(i, i+1)$ and $(j-1, j)$ must be a part of $M_{i,j}$ (as otherwise points i and j would be endpoints of two different diagonals from $M_{i,j}$, neither of which is (i, j)), which is not allowed by the requirement that the region containing (i, j) has at most one other bounding diagonal). If $(i, i+1) \in M_{i,j}$, then $M_{i,j}$ can be constructed from $M_{i+2,j}$ and the edge $(i, i + 1)$. Similarly, if $(j - 1, j) \in M_{i,j}$, then we can get $M_{i,j}$ as $M_{i,j-2}$ plus the edge $(j - 1, j)$.

Since these problems have optimal substructure, we can apply dynamic programming to solve them.

If $bn(M_{i,j})$ is saved into $S[i,j]$, the following recurrent formula can be used to calculate the solution to MATCHING(i,j) for all feasible pairs (i,j),

$$S[i,j] = \min \begin{cases} \max\{S[i+1,j-1], |v_i v_j|\} & \text{(1a)} \\ \max\{S[i+2,j], |v_i v_{i+1}|\} & \text{(1b)} \\ \max\{S[i,j-2], |v_{j-1} v_j|\}. & \text{(1c)} \end{cases}$$

Initially, we set $S[i,i] = 0$, for all i, and then we fill values in S in order of increasing $j-i$, so that all subproblems are already solved when needed.

Beside the value of a solution to MATCHING(i,j), it is going to be useful to determine if pair (i,j) is necessary for constructing $M_{i,j}$, i.e. we want to know do all solutions to MATCHING(i,j) contain (i,j). If that is true then we call such a pair *necessary*. This can be easily incorporated into the calculation of $S[i,j]$. Namely, if case (1a) is the only one achieving minimum among cases (1a), (1b) and (1c), we set $necessary(i,j)$ to $\top$, otherwise we set it to $\bot$.

We have $O(n^2)$ subproblems, each of which takes $O(1)$ time to be calculated. Hence, all calculations together require $O(n^2)$ time and the same amount of space. Note that we calculated only the values of solutions to all subproblems, but an actual matching can be easily reconstructed in linear time from the data in S.

4 Finding bottleneck matching

As we concluded earlier, there is a bottleneck matching of P having either at most one cascade, or exactly three cascades. An optimal matching with at most one cascade can be found easily from calculated solutions to subproblems. We just find the minimum of all $S[i+1,i]$, and take any $M_{i+1,i}$ that achieves it. This step takes only linear time.

Next, we focus on finding an optimal matching among all matchings with exactly three cascades (denoted by 3-*cascade matchings* in the following text).

Any three distinct points i, j and k, where (i,j), $(j+1,k)$ and $(k+1,i-1)$ are feasible pairs, can be used to construct a 3-cascade matching by simply taking a union of $M_{i,j}$, $M_{j+1,k}$ and $M_{k+1,i-1}$. To find the best one we could run through all possible triplets (i,j,k) and see which one minimizes $\max\{S[i,j], S[j+1,k], S[k+1,i-1]\}$. However, that requires $O(n^3)$ time, and thus is not suitable, since our goal is to design a faster algorithm. Our approach is to show that instead of looking at all (i,j) pairs, it is enough to select (i,j) from a set of linear size, which would reduce the search space to quadratic number of possibilities, so the search would take only $O(n^2)$ time.

Next, we prove a couple of simple statements about 3-cascade matchings. In 3-cascade matching, let us call the three diagonals bounding the single 3-bounded region the *inner* diagonals.

Lemma 3 *If there is no bottleneck matching with at most one cascade, then there is a bottleneck* 3*-cascade matching whose every inner diagonal is necessary.*

We say that (i,j) is a *candidate* diagonal, if it is a necessary diagonal and $\tau(i,j) \leq 2\pi/3$.

Lemma 4 *If there is no bottleneck matching with at most one cascade, then there is a* 3*-cascade bottleneck matching M, such that at least one inner diagonal of M is a candidate diagonal.*

Let us now look at a candidate diagonal (i,j), and examine the position of points $\{i+1,\ldots,j-1\}$ relative to it. We construct the circular arc h on the right side of the directed line $v_i v_j$, from which the line segment $v_i v_j$ subtends an angle of $\pi/3$, see Figure 3. We denote the midpoint of h with A. Points v_i, A and v_j form an equilateral triangle, hence we are able to construct the arc a^- between A and v_i with the center in v_j, and the arc a^+ between A and v_j with the center in v_i. These arcs define three areas: Π^-, bounded by h and a^-, Π^+, bounded by h and a^+, and Π^0, bounded by a^-, a^+ and the line segment $v_i v_j$, all depicted in Figure 3. Using these definitions we state the following lemma.

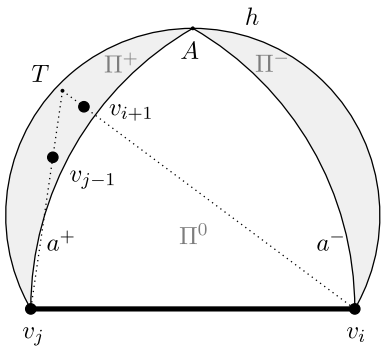


Figure 3: Points $v_{i+1},\ldots,v_{j-1}$ all lie inside either Π^- or Π^+.

Lemma 5 *If (i,j) is a candidate diagonal, then points $v_{i+1},\ldots,v_{j-1}$ either all belong to Π^- or all belong to Π^+.*

With $\Pi^-(i,j)$ and $\Pi^+(i,j)$ we respectively denote areas Π^- and Π^+ corresponding to the candidate diagonal (i,j).

Two possibilities for a candidate diagonal (i,j) provided by Lemma 5 bring forth a concept of *polarity*. If points $\{i+1,\ldots,j-1\}$ lie in $\Pi^-(i,j)$ we say that candidate diagonal (i,j) has *negative polarity* and has i as its *pole*. Otherwise, if these points lie in $\Pi^+(i,j)$, we say that (i,j) has *positive polarity* and pole in j.

We arrive at the crucial observation, which will enable us to limit the search space of the algorithm.

Lemma 6 *No two candidate diagonals of the same polarity can have the same point as a pole.*

A simple corollary of Lemma 6 is that there is at most linear number of candidate diagonals.

Lemma 7 *There are $O(n)$ candidate diagonals.*

Finally, we combine our findings from Lemma 4 and Lemma 7, as described in the beginning of Section 4, to construct Algorithm 1.

Algorithm 1 Bottleneck Matching

```
Calculate S[i,j] and necessary(i,j) for all feasible
(i,j) pairs, as described in Section 3.
best ← min{S[i+1,i] : i ∈ {0,...,n−1}}
for all feasible (i,j) do
    if necessary(i,j) and τ(i,j) ≤ 2π/3 then
        for k ∈ {j+1,...,i−1} such that (j+1,k)
        is feasible do
            best ← min{best, max{S[i,j], S[j+1,k],
                                 S[k+1,i−1]}}
        end for
    end if
end for
```

Theorem 8 *Algorithm 1 finds the value of bottleneck matching in $O(n^2)$ time.*

Proof. The first step, calculating $S[i,j]$ and $necessary(i,j)$ for all (i,j) pairs, is done in $O(n^2)$ time, as described in Section 3. The second step finds the minimal value of all matchings with at most one cascade in $O(n)$ time.

The rest of the algorithm finds the minimal value of all 3-cascade matchings. Lemma 4 tells us that there is a bottleneck matching among 3-cascade matchings with one inner diagonal being a candidate diagonal, so the algorithm searches through all such matchings. We first fix the candidate diagonal (i,j) and then enter the inner for-loop, where we search for an optimal 3-cascade matching having (i,j) as an inner diagonal. Although the outer for-loop is executed $O(n^2)$ times, Lemma 7 guarantees that the if-block is entered only $O(n)$ times. The inner for-loop splits $\{j+1,\ldots,i-1\}$ in two parts, $\{j+1,\ldots,k\}$ and $\{k+1,\ldots,i-1\}$, which together with $\{i,\ldots,j\}$ make three parts, each to be matched with at most one cascade. We already know the values of optimal solutions for these three subproblems, so we combine them and check if we get a better overall value. At the end, the minimum value of examined matchings is contained in *best*, and that has to be the value of a bottleneck matching, since we surely examined at least one bottleneck matching. □

Algorithm 1 gives only the value of a bottleneck matching, however, it is easy to reconstruct an actual bottleneck matching by reconstructing matchings for subproblems that led to the minimum value. This reconstruction can be done in linear time.

References

[1] A. K. Abu-Affash, A. Biniaz, P. Carmi, A. Maheshwari, and M. Smid. Approximating the bottleneck plane perfect matching of a point set. *Computational Geometry*, 48(9):718 – 731, 2015.

[2] A. K. Abu-Affash, P. Carmi, M. J. Katz, and Y. Trabelsi. Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47(3):447–457, 2014.

[3] O. Aichholzer, S. Bereg, A. Dumitrescu, A. García, C. Huemer, F. Hurtado, M. Kano, A. Márquez, D. Rappaport, S. Smorodinsky, D. Souvaine, J. Urrutia, and D. R. Wood. Compatible geometric matchings. *Computational Geometry*, 42(6):617–626, 2009.

[4] O. Aichholzer, S. Cabello, R. Fabila-Monroy, D. Flores-Penaloza, T. Hackl, C. Huemer, F. Hurtado, and D. R. Wood. Edge-removal and non-crossing configurations in geometric graphs. *Discrete Mathematics and Theoretical Computer Science*, 12(1):75–86, 2010.

[5] N. Alon, S. Rajagopalan, and S. Suri. Long non-crossing configurations in the plane. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 257–263. ACM, 1993.

[6] G. Aloupis, E. M. Arkin, D. Bremner, E. D. Demaine, S. P. Fekete, B. Kouhestani, and J. S. Mitchell. Matching regions in the plane using non-crossing segments. EGC, 2015.

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

[8] A. Biniaz, A. Maheshwari, and M. Smid. Bottleneck bichromatic plane matching of points. Canadian Conference on Computational Geometry, 2014.

[9] J. G. Carlsson, B. Armbruster, H. Bellam, and R. Saladi. A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry and Applications*, to appear.

[10] M.-S. Chang, C. Y. Tang, and R. C. T. Lee. Solving the euclidean bottleneck matching problem by k-relative neighborhood graphs. *Algorithmica*, 8(1-6):177–194, 1992.

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

[12] A. Efrat and M. J. Katz. Computing euclidean bottleneck matchings in higher dimensions. *Information processing letters*, 75(4):169–174, 2000.

[13] J. Kratochvíl and T. Ueckerdt. Non-crossing connectors in the plane. In *Theory and Applications of Models of Computation*, volume 7876 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2013.

Bottleneck Matchings and Hamiltonian Cycles in Higher-Order Gabriel Graphs*

Ahmad Biniaz† Anil Maheshwari† Michiel Smid†

Abstract

Given a set P of n points in the plane, the order-k Gabriel graph on P, denoted by k-GG, has an edge between two points p and q if and only if the closed disk with diameter pq contains at most k points of P, excluding p and q. It is known that 10-GG contains a Euclidean bottleneck matching of P, while 8-GG may not contain such a matching. We answer the following question in the affirmative: does 9-GG contain any Euclidean bottleneck matching of P?

It is also known that 10-GG contains a Euclidean bottleneck Hamiltonian cycle of P, while 5-GG may not contain such a cycle. We improve the lower bound and show that 7-GG may not contain any Euclidean bottleneck Hamiltonian cycle of P.

1 Introduction

Let P be a set of n points in the plane. For any two points $p, q \in P$, let $D[p, q]$ denote the closed disk that has the line segment $\overline{pq}$ as diameter. Let $|pq|$ be the Euclidean distance between p and q. The *Gabriel graph* on P, denoted by $GG(P)$, is a geometric graph that has an edge between two points p and q if and only if $D[p, q]$ does not contain any point of $P \setminus \{p, q\}$. Gabriel graphs were introduced by Gabriel and Sokal [6] and can be computed in $O(n \log n)$ time [8]. Every Gabriel graph has at most $3n - 8$ edges, for $n \geq 5$, and this bound is tight [8].

The *order-k Gabriel graph* on P, denoted by k-GG, is the geometric graph that has an edge between two points p and q if and only if $D[p, q]$ contains at most k points of $P \setminus \{p, q\}$. Thus, the Gabriel graph, $GG(P)$, corresponds to 0-GG. Su and Chang [9] showed that k-GG can be constructed in $O(k^2 n \log n)$ time and contains $O(k(n - k))$ edges. For two points $p, q \in P$, the *lune* of p and q, denoted by $L(p, q)$, is defined as the intersection of the two open disks of radius $|pq|$ centered at p and q. The *order-k Relative Neighborhood Graph* on P, denoted by k-RNG, is the geometric graph that has an edge (p, q) if and only if $L(p, q)$ contains at most k points of P. Note that k-RNG on P is a subgraph of k-GG on P.

A *matching* in a graph G is a set of edges without common vertices. A *perfect matching* is a matching that matches all the vertices of G. A *Hamiltonian cycle* in G is a cycle that visits each vertex of G exactly once. In the case when G is an edge-weighted graph, a *bottleneck matching* is defined to be a perfect matching in G, in which the weight of the maximum-weight edge is minimized. Moreover, a *bottleneck Hamiltonian cycle* is a Hamiltonian cycle in G, in which the weight of the maximum-weight edge is minimized. For a point set P, a *Euclidean bottleneck matching* is a perfect matching in the complete graph with vertex set P that minimizes the longest edge; the weight of an edge is defined to be the Euclidean distance between its two endpoints. Similarly, a *Euclidean bottleneck Hamiltonian cycle* is a Hamiltonian cycle that minimizes the longest edge.

Chang et al. [4] proved that a Euclidean bottleneck matching of P is contained in 16-RNG.[1] This implies that 16-GG contains a Euclidean bottleneck matching. In [2] the authors improved the bound for the latter graphs by showing that 10-GG contains a Euclidean bottleneck matching. They also show that 8-GG may not have any Euclidean bottleneck matching. They asked if 9-GG contains any Euclidean bottleneck matching. In Section 2, we answer this question in the affirmative.

Theorem 1 *For every point set P, 9-GG contains a Euclidean bottleneck matching of P.*

Chang et al. [3] proved that a Euclidean bottleneck Hamiltonian cycle of P is contained in 19-RNG, which implies that 19-GG contains a Euclidean bottleneck Hamiltonian cycle. Abellanas et al. [1] improved the bound by showing that 15-GG contains a Euclidean bottleneck Hamiltonian cycle. Kaiser et al. [7] improved the bound further by showing that 10-GG contains a Euclidean bottleneck Hamiltonian cycle. They also provide an example which shows that 5-GG may not contain any Euclidean bottleneck Hamiltonian cycle. In Section 3, we improve the lower bound to 7 and prove the following proposition.

Proposition 1 *There exist point sets P such that 7-GG does not contain any Euclidean bottleneck Hamiltonian cycle of P.*

*Research supported by NSERC.

†Carleton University, Ottawa, Canada.

[1] They defined k-RNG to have an edge (p, q) if and only if $L(p, q)$ contains at most $k - 1$ points of P.

Therefore, it remains open to decide whether or not 8-*GG* or 9-*GG* contains a Euclidean bottleneck Hamiltonian cycle.

2 Proof of Theorem 1

In this section we prove Theorem 1. The proofs for Lemmas 2 and 3 are similar to the proofs in [4] which are adjusted for Gabriel graphs. The proof of Lemma 4 is based on a similar technique that is used in [7] for the Hamiltonicity of Gabriel graphs.

Let $\mathcal{M}$ be the set of all perfect matchings of the complete graph with vertex set P. For a matching $M \in \mathcal{M}$ we define the *weight sequence* of M, $\mathrm{WS}(M)$, as the sequence containing the weights of the edges of M in non-increasing order. A matching M_1 is said to be less than a matching M_2 if $\mathrm{WS}(M_1)$ is lexicographically smaller than $\mathrm{WS}(M_2)$. We define a total order on the elements of $\mathcal{M}$ by their weight sequence. If two elements have exactly the same weight sequence, break ties arbitrarily to get a total order.

Let $M^* = \{(a_1, b_1), \ldots, (a_{\frac{n}{2}}, b_{\frac{n}{2}})\}$ be a matching in $\mathcal{M}$ with minimum weight sequence. Observe that M^* is a Euclidean bottleneck matching for P. In order to prove Theorem 1, we will show that all edges of M^* are in 9-*GG*. Consider any edge (a, b) in M^*. If $D[a, b]$ contains no point of $P \setminus \{a, b\}$, then (a, b) is an edge of 9-*GG*. Suppose that $D[a, b]$ contains k points of $P \setminus \{a, b\}$. We are going to prove that $k \leq 9$. Let $R = \{r_1, r_2, \ldots, r_k\}$ be the set of points of $P \setminus \{a, b\}$ that are in $D[a, b]$. Let $S = \{s_1, s_2, \ldots, s_k\}$ represent the points for which $(r_i, s_i) \in M^*$.

Without loss of generality, we assume that $D[a, b]$ has diameter 1 and is centered at the origin $o = (0, 0)$, and $a = (-0.5, 0)$ and $b = (0.5, 0)$. For any point p in the plane, let $\|p\|$ denote the distance of p from o. Note that $|ab| = 1$, and for any point $x \in D[a, b] \setminus \{a, b\}$ we have $\max\{|xa|, |xb|\} < 1$.

Lemma 2 *For each point $s_i \in S$, $\min\{|s_i a|, |s_i b|\} \geq 1$.*

Proof. The proof is by contradiction; suppose that $|s_i a| < 1$. Let M be the perfect matching obtained from M^* by deleting $\{(a, b), (r_i, s_i)\}$ and adding $\{(s_i, a), (r_i, b)\}$. The lengths of the two new edges are smaller than 1, and hence both (s_i, a) and (r_i, b) are shorter than (a, b). Thus, $\mathrm{WS}(M) <_{\mathrm{lex}} \mathrm{WS}(M^*)$, which contradicts the minimality of M^*. □

As a corollary of Lemma 2, R and S are disjoint.

Lemma 3 *For each pair of points $s_i, s_j \in S$, $|s_i s_j| \geq \max\{|r_i s_i|, |r_j s_j|, 1\}$.*

Proof. The proof is by contradiction; suppose that $|s_i s_j| < \max\{|r_i s_i|, |r_j s_j|, 1\}$. Let M be the perfect matching obtained from M^* by deleting $\{(a, b), (r_i, s_i), (r_j, s_j)\}$ and adding $\{(a, r_i), (b, r_j), (s_i, s_j)\}$. Note that $\max\{|ar_i|, |br_j|, |s_i s_j|\} < \max\{|r_i s_i|, |r_j s_j|, |ab|\}$. Thus, $\mathrm{WS}(M) <_{\mathrm{lex}} \mathrm{WS}(M^*)$, which contradicts the minimality of M^*. □

Let $C(x, r)$ (resp. $D(x, r)$) be the circle (resp. closed disk) of radius r that is centered at a point x in the plane. For $i \in \{1, \ldots, k\}$, let s'_i be the intersection point between $C(o, 1.5)$ and the ray with origin at o passing through s_i. Let the point p_i be s_i, if $\|s_i\| < 1.5$, and s'_i, otherwise. See Figure 1. Let $S' = \{a, b, p_1, \ldots, p_k\}$.

Observation 1 *Let s_j be a point in S, where $\|s_j\| \geq 1.5$. Then, the disk $D(s_j, \|s_j\| - 0.5)$ is contained in the disk $D(s_j, |s_j r_j|)$. Moreover, the disk $D(p_j, 1)$ is contained in the disk $D(s_j, \|s_j\| - 0.5)$. See Figure 1.*

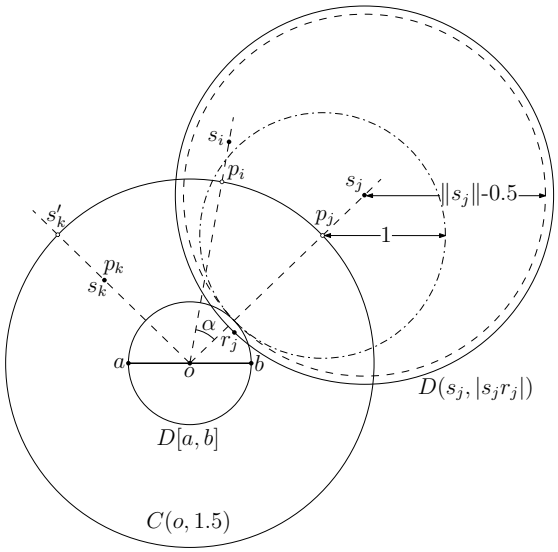


Figure 1: Proof of Lemma 4; $p_i = s'_i$, $p_j = s'_j$, and $p_k = s_k$.

Lemma 4 *The distance between any pair of points in S' is at least 1.*

Proof. Let x and y be two points in S'. We are going to prove that $|xy| \geq 1$. We distinguish between the following three cases.

- $\{x, y\} = \{a, b\}$. In this case the claim is trivial.
- $x \in \{a, b\}, y \in \{p_1, \ldots, p_k\}$. If $\|y\| = 1.5$, then y is on $C(o, 1.5)$, and hence $|xy| \geq 1$. If $\|y\| < 1.5$, then y is a point in S. Therefore, by Lemma 2, $|xy| \geq 1$.
- $x, y \in \{p_1, \ldots, p_k\}$. Without loss of generality assume $x = p_i$ and $y = p_j$, where $1 \leq i < j \leq k$. We differentiate between three cases:

Case (i): $\|p_i\| < 1.5$ and $\|p_j\| < 1.5$. In this case p_i and p_j are two points in S. Therefore, by Lemma 3, $|p_ip_j| \geq 1$.

Case (ii): $\|p_i\| < 1.5$ and $\|p_j\| = 1.5$. In this case p_i is a point in S. By Observation 1, the disk $D(p_j, 1)$ is contained in the disk $D(s_j, |s_jr_j|)$, and by Lemma 3, p_i is not in the interior of $D(s_j, |s_jr_j|)$. Therefore, p_i is not in the interior of $D(p_j, 1)$, which implies that $|p_ip_j| \geq 1$.

Case (iii): $\|p_i\| = 1.5$ and $\|p_j\| = 1.5$. In this case $\|s_i\| \geq 1.5$ and $\|s_j\| \geq 1.5$. Without loss of generality assume $\|s_i\| \leq \|s_j\|$. For the sake of contradiction assume that $|p_ip_j| < 1$; see Figure 1. Then, for the angle $\alpha = \angle s_ios_j$ we have $\sin(\alpha/2) < \frac{1}{3}$. Then, $\cos(\alpha) = 1 - 2\sin^2(\alpha/2) > \frac{7}{9}$. By the law of cosines in the triangle $\triangle s_ios_j$, we have

$$|s_is_j|^2 < \|s_i\|^2 + \|s_j\|^2 - \frac{14}{9}\|s_i\|\|s_j\|. \qquad (1)$$

By Observation 1, the disk $D(s_j, \|s_j\| - 0.5)$ is contained in the disk $D(s_j, |s_jr_j|)$, and by Lemma 3, s_i is not in the interior of $D(s_j, |s_jr_j|)$. Therefore, s_i is not in the interior of $D(s_j, \|s_j\| - 0.5)$. Thus, $|s_is_j| \geq \|s_j\| - 0.5$. In combination with Inequality (1), this implies

$$\|s_j\| \left(\frac{14}{9}\|s_i\| - 1\right) < \|s_i\|^2 - \frac{1}{4}. \qquad (2)$$

In combination with the assumption $\|s_i\| \leq \|s_j\|$, Inequality (2) implies

$$\frac{5}{9}\|s_i\|^2 - \|s_i\| + \frac{1}{4} < 0,$$

i.e.,

$$\frac{5}{9}\left(\|s_i\| - \frac{3}{10}\right)\left(\|s_i\| - \frac{3}{2}\right) < 0.$$

This is a contradiction, because, since $\|s_i\| \geq 1.5$, the left-hand side is non-negative. Thus $|p_ip_j| \geq 1$, which completes the proof of the lemma.

□

By Lemma 4, the points in S' have mutual distance at least 1. Moreover, the points in S' lie in $D(o, 1.5)$. Fodor [5] proved that the smallest circle which contains 12 points with mutual distances at least 1 has radius 1.5148. Therefore, S' contains at most 11 points. Since $a, b \in S'$, this implies that $k \leq 9$. Therefore, S, and consequently R, contains at most 9 points. Thus, (a, b) is an edge in 9-GG. This completes the proof of Theorem 1.

3 Proof of Proposition 1

In this section we prove Proposition 1. We show that for some point sets P, 7-GG does not contain any Euclidean bottleneck Hamiltonian cycle of P.

Figure 2 shows a configuration of a multiset $P = \{a, b, x, r_1, \ldots, r_8, s_1, \ldots, s_7\}$ of 26 points, where s_5 is repeated nine times. The closed disk $D[a, b]$ is centered at o and has diameter one, i.e., $|ab| = 1$. $D[a, b]$ contains all 8 points of the set $R = \{r_1, \ldots, r_8\}$; these points lie on the circle with radius $\frac{1}{2} - \epsilon$ that is centered at o; all points of R are in the interior of $D[a, b]$. Let $S = \{s_1, \ldots, s_7\}$ be the multiset of 15 points, where s_5 is repeated nine times. The red circles have radius 1 and are centered at points in S. Each point in S is connected to its first and second closest point (the black edges in Figure 2). Let B the chain formed by these edges. Note that r_1 and r_8 are the endpoints of B. Specifically, $|r_1s_1| = |r_8s_7| = 1$, and for each point r_i, where $2 \leq i \leq 7$, $|s_ia| > 1$, $|s_ib| > 1$, $|s_ix| > 1$, and $|r_is_{i-1}| = |r_is_i| = 1$ (here by s_5 we mean the first and last endpoints of the chain defined by points labeled s_5). Consider the Hamiltonian cycle $H = B \cup \{(r_1, a), (a, b), (b, x), (x, r_8)\}$. The longest edge in H has length 1. Therefore, the length of the longest edge in any bottleneck Hamiltonian cycle for P is at most 1. In the rest we will show—by contradiction—that any bottleneck Hamiltonian cycle of P contains (a, b). Since in B each point of S is connected to its first and second closest point, every bottleneck Hamiltonian cycle of P contains B, because otherwise, one of the points in S should be connected to a point that is farther than its second closest point, and hence that edge is longer than 1. Now we consider possible ways to construct a bottleneck Hamiltonian cycle, say H^*, using the edges in B and the points a, b, x. Assume $(a, b) \notin H^*$. Then, in H^*, a is connected to two points in $\{r_1, r_8, x\}$. We differentiate between two cases:

- $(a, x) \in H^*$. In this case $|ax| > 1$, and hence the longest edge in H^* is longer than 1, which is a contradiction.
- $(a, x) \notin H^*$. In this case $(a, r_1) \in H^*$ and $(a, r_8) \in H^*$. This means that H^* does not contain x and b, which is a contradiction.

Therefore, we conclude that H^*, and consequently any bottleneck Hamiltonian cycle of P, contains (a, b). Since $D[a, b]$ contains 8 points of $P \setminus \{a, b\}$, $(a, b) \notin$ 7-GG. Therefore 7-GG does not contain any Euclidean bottleneck Hamiltonian cycle of P.

4 Conclusion

We considered the inclusion of a Euclidean bottleneck matching and a Euclidean bottleneck Hamiltonian cycle of a point set P in higher order Gabriel graphs. It

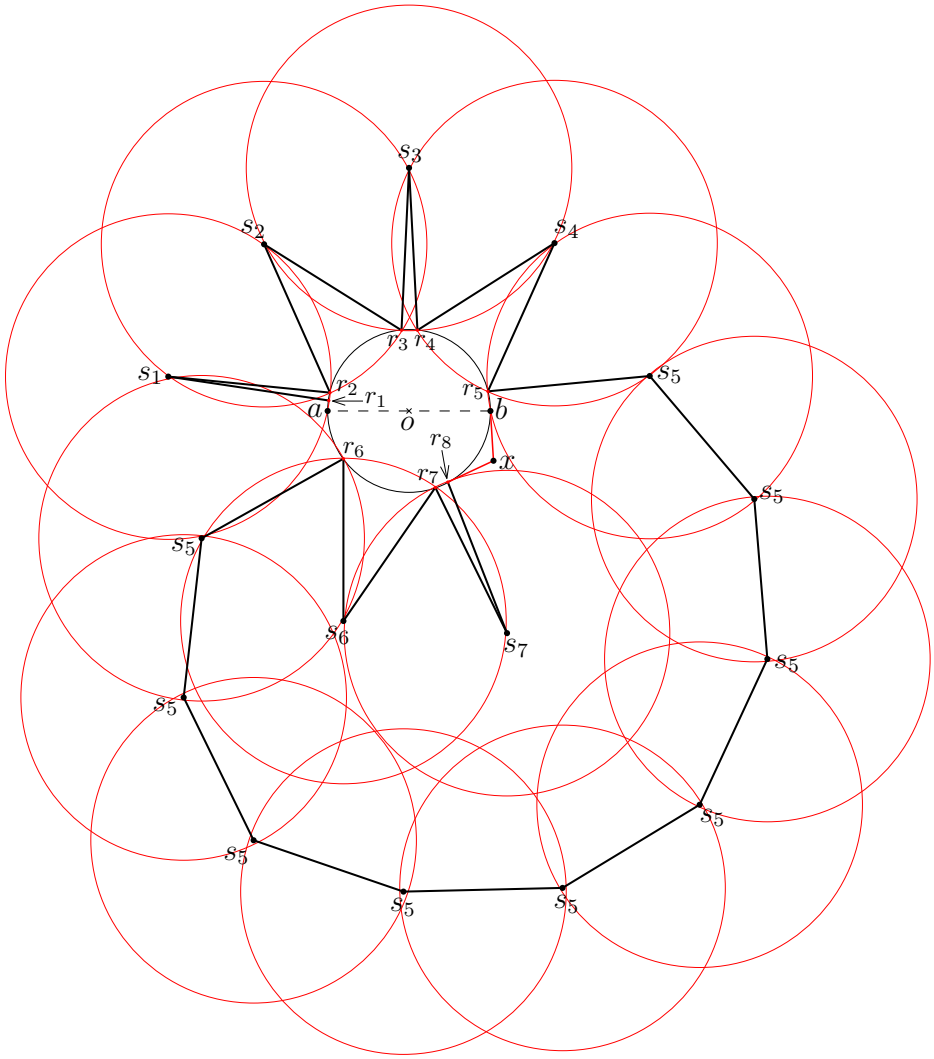


Figure 2: Proof of Proposition 1. The bold-black edges belong to B. $D[a, b]$ contains 8 points.

is known that 10-GG contains a bottleneck matching and a bottleneck Hamiltonian cycle of P. We proved that 9-GG contains a bottleneck matching of P and 7-GG may not contain any bottleneck Hamiltonian cycle of P. It remains open to decide if 8-GG or 9-GG contains any bottleneck Hamiltonian cycle of P.

References

[1] M. Abellanas, P. Bose, J. García-López, F. Hurtado, C. M. Nicolás, and P. Ramos. On structural and graph theoretic properties of higher order Delaunay graphs. *Int. J. Comput. Geometry Appl.*, 19(6):595–615, 2009.

[2] A. Biniaz, A. Maheshwari, and M. Smid. Matchings in higher-order Gabriel graphs. *Theor. Comput. Sci.*, 596:67–78, 2015.

[3] M.-S. Chang, C. Y. Tang, and R. C. T. Lee. 20-relative neighborhood graphs are Hamiltonian. *Journal of Graph Theory*, 15(5):543–557, 1991.

[4] M.-S. Chang, C. Y. Tang, and R. C. T. Lee. Solving the Euclidean bottleneck matching problem by k-relative neighborhood graphs. *Algorithmica*, 8(3):177–194, 1992.

[5] F. Fodor. The densest packing of 12 congruent circles in a circle. *Beiträge Algebra Geom*, 41:401–409, 2000.

[6] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, 1969.

[7] T. Kaiser, M. Saumell, and N. V. Cleemput. 10-Gabriel graphs are Hamiltonian. *Inf. Process. Lett.*, 115(11):877–881, 2015.

[8] D. W. Matula and R. R. Sokal. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12(3):205–222, 1980.

[9] T.-H. Su and R.-C. Chang. The k-Gabriel graphs and their applications. In *SIGAL International Symposium on Algorithms*, pages 66–75, 1990.

Dynamic Connectivity for Unit Disk Graphs*

Haim Kaplan† Wolfgang Mulzer‡ Liam Roditty§ Paul Seiferth‡

Abstract

Let $S \subset \mathbb{R}^2$ be a set of point sites. The *unit disk graph* $\mathrm{UD}(S)$ of S has vertex set S and an edge between two sites s, t if and only if $|st| \leq 1$.

We present a data structure that maintains the connected components of $\mathrm{UD}(S)$ when S changes dynamically. It takes $O(\log^2 n)$ time to insert or delete a site in S and $O(\log n / \log\log n)$ time to determine if two sites are in the same connected component. Here, n is the maximum size of S at any time. A simple variant improves the update time to $O(\log n \log\log n)$ at the cost of a slightly increased query time of $O(\log n)$.

1 Introduction

Computing the connected components of a graph G is one of the most fundamental problems in algorithmic graph theory. When G is static, several classic solutions exist, e.g., BFS or DFS. However, if G can change dynamically, the problem becomes much more challenging. In this case, we would like a data structure for *connectivity queries*: given two vertices s and t, are s and t in the same connected component of G? Additionally, we would like to be able to insert and delete edges or singleton vertices. For general graphs, there is the following result due to Holm et al. [8].

Theorem 1 (Holm et al., Theorem 3) *Let G be a graph with n vertices. There is a deterministic data structure such that edge insertions or deletions in G take amortized time $O(\log^2 n)$, and connectivity queries take worst-case time $O(\log n / \log\log n)$.*

Even though Theorem 1 assumes n to be fixed, we can use a standard rebuilding method to support vertex insertion and deletion within the same amortized time bounds, by rebuilding the data structure whenever the number of vertices changes by a factor of 2. For planar graphs, Eppstein et al. achieved $O(\log n)$ time for both updates and queries [7].

However, the model of edge insertions and deletions may be too restrictive. For example, one natural situation where more powerful operations are needed occurs in *unit disk graphs*. Let $S \subset \mathbb{R}^2$ be a set of point sites. The *unit disk graph* $\mathrm{UD}(S)$ of S has vertex set S and an edge between two sites $s, t \in S$ if and only if the Euclidean distance $|st|$ is at most 1. Now, we want to maintain the connected components of $\mathrm{UD}(S)$ as the *vertex set* S changes dynamically. In this case, a single update may change the graph quite dramatically, since one site may have many incident edges. Nevertheless, Chan et al. [5] observed that by combining known results one can derive a data structure with update time $O(\log^{10} n)$ and query time $O(\log n / \log\log n)$. The construction is as follows (see Figure 1): ① let T be the Euclidean minimum spanning tree (EMST) of S. If we remove all edges with length larger than 1 from T, the resulting forest F is a spanning forest for $\mathrm{UD}(S)$. Thus, to maintain the components of $\mathrm{UD}(S)$, it suffices to maintain the components of F. We create data structure D of Holm et al. to maintain F. Since the EMST has maximum degree 6, inserting or deleting a site from S changes $O(1)$ edges in T. Suppose we can efficiently find the set E of edges that change during an update. Then, we can update the components in F through $O(1)$ updates in D, taking all edges in E of length at most 1. ② To find E, we need to dynamically maintain the EMST T when S changes. This can be done using a technique of Agarwal et al. that reduces the problem to several instances of the *dynamic bichromatic closest pair problem* (DBCP), with an overhead of $O(\log^2 n)$ in the update time [1]. ③ Eppstein showed that the DBCP problem can in turn be solved through a reduction to several instances of the dynamic nearest neighbor problem (DNN) for points in the plane [6]. Again, we incur another $O(\log^2 n)$ factor as overhead in the update time. Using Chan's DNN structure [4] with amortized expected update time $O(\log^6 n)$, we get a total update time of $O(\log^{10} n)$. We can use D to answer queries in $O(\log n / \log\log n)$ time.

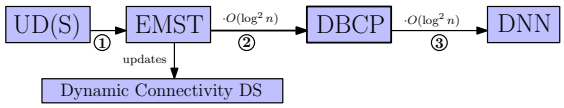


Figure 1: A solution with $O(\log^{10} n)$ update time.

Our Results. We improve the previous result by following a similar approach, but in every step we use a method more specifically tailored to unit disks. Instead of the EMST in ①, we use a much simpler graph

*Supported by GIF project 1161&DFG project MU/3501-1.

†Tel Aviv University, Israel. haimk@post.tau.ac.il

‡Institut für Informatik, Freie Universität Berlin, Germany {mulzer,pseiferth}@inf.fu-berlin.de

§Bar Ilan University, Israel. liamr@macs.biu.ac.il

on grid cells that also captures the connectivity of UD(S). Then we can avoid the $O(\log^2 n)$ overhead in ② and ③ and substitute the DNN data structure by a *dynamic lower envelope* (DLE) structure for pseudolines in $\mathbb{R}^2$. In Section 2 we review suitable DLE structures and their properties. In Section 3 we prove our first main theorem:

Theorem 2 *There is a dynamic connectivity structure for unit disk graphs such that the insertion or deletion of a site takes amortized time $O(\log^2 n)$ and a connectivity query takes worst-case time $O(\log n/\log\log n)$, where n is the maximum number of sites at any time.*

In Section 4, we use a grid-based *planar* graph to represent the connectivity of UD(S). Then we can replace Theorem 1 by the result for planar graphs by Eppstein et al. Updates now take $O(\log n \log\log n)$ time, but the query time slightly increases to $O(\log n)$.

2 Dynamic Lower Envelopes

Let L be a set of *pseudolines* in the plane, i.e., each element of L is a simple continuous curve and any two distinct curves in L intersect in exactly one point. The *lower envelope* of L is the pointwise minimum of the graphs of the curves in L. In Section 3 we need to dynamically maintain the lower envelope of L. Overmars and van Leeuwen show how to maintain the lower envelope of a set of lines with update time $O(\log^2 n)$ such that vertical ray shooting queries can be answered in $O(\log^2 n)$ time [10]. Chan improves this to $O(\log^{1+\varepsilon})$ for updates and queries [3]. Using the kinetic heap structure of Kaplan et al. [9] one can obtain $O(\log n \log\log n)$. Brodal and Jacob showed that the optimal bound $O(\log n)$ can be achieved [2]. Except for the last result, one can verify that all these approaches also work with pseudolines; they only need a total ordering of the lines along the lower envelope.

Lemma 3 *Let L be a dynamic set of at most n pseudolines. We can maintain the lower envelope of L with $O(\log n \log\log n)$ amortized update time and $O(\log n)$ amortized query time.*

Remark. The applicability of the result by Brodal and Jacob [2] is not clear to us, and poses an interesting challenge for further investigation.

3 The Data Structure

Let $S \subset \mathbb{R}^2$ be a set of sites. We define an *auxiliary graph* G that represents the connectivity of UD(S). The vertices of G are cells of a grid. To see if two cells form an edge, we maintain a bichromatic matching of the sites in the grid cells. This matching is updated with the help of two DLE data structures.

The Grid Graph (new ①). Let $\mathcal{G}$ be a planar grid whose cells are disjoint axis-aligned squares with diameter 1. For any grid cell $\sigma \in \mathcal{G}$, the sites $\sigma \cap S$ induce a clique in UD(S). For $S \subset \mathbb{R}^2$, we define a graph G whose vertices are the *non-empty* cells $\sigma \in \mathcal{G}$, i.e., the cells with $\sigma \cap S \neq \emptyset$. The *neighborhood* $N(\sigma)$ of a cell $\sigma \in \mathcal{G}$ is the 5×5 block of cells in $\mathcal{G}$ with σ in the center. We call two cells *neighboring* if they are in each other's neighborhood. The endpoints of any edge in UD(S) must lie in neighboring cells. To obtain the edges of G, we connect every pair of distinct neighboring grid cells that contain the endpoints of an edge in UD(S). By construction, and since the sites inside each cell form a clique, the connectivity between two sites s, t in UD(S) is the same as for the corresponding cells in G.

Lemma 4 *Let $s, t \in S$ be two sites and let σ and τ be the cells in $\mathcal{G}$ that contain s and t, respectively. There is an s-t path in* UD(S) *if and only if there is a σ-τ path in G.*

We build the data structure from Theorem 1 for G. When a site s is inserted into or deleted from S, only $O(1)$ edges in G change, since only the neighborhood of the cell of s is affected. Thus, once the set E of changing edges is determined, we can update G in time $O(\log^2 n)$, by Theorem 1.

Finding the Edges E (new ②). It remains to find the edges E of G that change when we update S. For this, we maintain for each pair of non-empty neighboring cells a *maximal bichromatic matching* (MBM) between their sites, similar to Eppstein's method [6]. Let $R \subseteq S$ and $B \subseteq S$ be two sets of sites. An MBM between R and B is a maximal set of vertex-disjoint edges in $(R \times B) \cap$ UD(S), the bipartite graph on $R \cup B$ consisting of all edges of UD(S) with one endpoint in R and one endpoint in B.

For each pair $\{\sigma, \tau\}$ of neighboring cells in $\mathcal{G}$, we build an MBM $M_{\{\sigma,\tau\}}$ for $R = \sigma \cap S$ and $B = \tau \cap S$. By definition, there is an edge between σ and τ in G if and only if $M_{\{\sigma,\tau\}}$ is not empty. When inserting or deleting a site s from S, we proceed as follows: let $\sigma \in \mathcal{G}$ be the cell with $s \in \sigma$. We go through all cells $\tau \in N(\sigma)$ and update the MBM $M_{\{\sigma,\tau\}}$ (by inserting or deleting s from the relevant set). If $M_{\{\sigma,\tau\}}$ becomes non-empty during an insertion or becomes empty during a deletion, we add the edge $\sigma\tau$ to E and mark it for insertion or deletion, respectively. We summarize this construction in the following lemma.

Lemma 5 *Suppose we can maintain an MBM for each pair of non-empty neighboring cells with update time $O(U(n))$, where n is the maximum number of sites. Then we can dynamically maintain the adjacency lists of G with update time $O(U(n))$.*

Dynamically Maintaining an MBM (new ③). Let $\sigma \neq \tau$ be two neighboring cells of $\mathcal{G}$, and let $R = \sigma \cap S$ and $B = \tau \cap S$. We show that an MBM between R and B can be efficiently maintained using two DLE structures for pseudolines. We fix a line ℓ that separates R and B. Since R, B are in two distinct grid cells, we can take a supporting line of one of the four boundaries of σ. We have the following lemma.

Lemma 6 *Let $R, B \subseteq S$ be two sets with a total of at most n sites, separated by a line ℓ. There exists a dynamic data structure that maintains an MBM for R and B with $O(\log n \log\log n)$ update time.*

Proof. We rotate and translate everything such that ℓ is the x-axis and all sites in R have positive x-coordinate. We consider the set U_R of unit disks with centers in R (see Figure 2). Then a site in B forms an edge with *some* site in R if and only if it is contained in the union of the disks in U_R. To detect this, we maintain the lower envelope of U_R. More precisely, consider the following set L_R of pseudolines: for each disk of U_R, take the arc that defines the lower part of the boundary of the disk and extend both ends straight upward to ∞. We build a data structure D_R

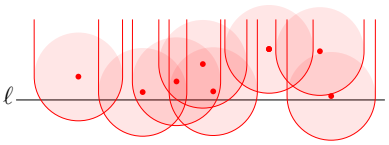


Figure 2: The set L_R induced by R.

for L_R according to Lemma 3. Analogously, we define a set of pseudolines L_B and a dynamic envelope structure D_B for B.

To maintain the MBM M, we store in D_R the currently unmatched sites of R, and in D_B the currently unmatched sites of B. When inserting a site r into R, we perform a vertical ray shooting query in D_B with r to get a pseudoline of L_B. Let $b \in B$ the site for that pseudoline. If $|rb| \leq 1$, we add the edge rb to M, and delete the pseudoline of b from D_B. Otherwise we insert the pseudoline of r into D_R. By construction, if there is an edge between r and an unmatched site in B, then there is also an edge between r and b. Hence, the insertion procedure correctly maintains an MBM. Now suppose we want to delete a site r from R. If r is unmatched, we delete the pseudoline corresponding to r from D_R. Otherwise, we remove the edge rb from M, and we reinsert b as above, looking for a new unmatched site in R for b. Updating B is analogous.

Inserting and deleting a site requires $O(1)$ insertions, deletions, or queries in D_R or D_B, so the lemma follows. □

To obtain Theorem 2, we combine Lemma 4,5, and 6.

4 Improving the Update Time

The bottleneck for the update time in Section 3 lies in the use of Theorem 1. We now define a planar graph G_p that is similar to the grid graph G: it represents the connectivity of UD(S) and an update of S changes $O(1)$ vertices and edges in G_p. These vertices and edges can be found in $O(1)$ time. Since G_p is planar, we can use the result of Eppstein et al. to maintain the connectivity of G_p with $O(\log n)$ amortized update and worst-case query time [7], giving the next theorem.

Theorem 7 *There is a dynamic connectivity structure for unit disk graphs such that insertion or deletion of a site takes amortized time $O(\log n \log\log n)$ and a connectivity query takes worst-case time $O(\log n)$, where n is the maximum number of sites at any time.*

The Planar Graph. Let $S \subset \mathbb{R}^2$ be a set of sites. For any pair of non-empty grid cells σ, τ, let $M_{\{\sigma,\tau\}}$ be the MBM as above. For any non-empty MBM $M_{\{\sigma,\tau\}}$, we pick an arbitrary edge $rb \in M_{\{\sigma,\tau\}}$ with $r \in \sigma$ and $b \in \tau$ as *representative edge*. Let $T \subseteq S$ be the set of sites incident to a representative edge. We use the unit disk graph UD(T) as basis for our planar graph G_p. If we contract in each grid cell σ the subgraph of UD(T) induced by $T \cap \sigma$ to a single vertex, we get the graph G from Section 3. Hence, by Lemma 4, UD(T) represents the connectivity of UD(S).

To get G_p from UD(T), we consider the straight line drawing of UD(T). For a crossing of two edges st and uv in UD(T), we add a new site x at the intersection and call x a *crossing site*. We remove st and uv and we add the four new edges sx, xt, ux, and xv. We repeat this operation until there are no more crossings in UD(T). This is a standard method for making unit disk graphs planar. The next lemma, due to Yan et al. [11], shows that it preserves connectivity.

Lemma 8 *Let ab and uv be edges in UD(T) that cross. Then a, b, u, and v are in the same connected component of UD($\{a, b, u, v\}$).*

Using Lemma 8 we now show that G_p has the same connectivity as UD(T). Thus, by Lemma 4, G_p represents the connectivity of UD(S).

Lemma 9 *Let $s, t \in T$ be two sites. Then s and t are connected in UD(T) if and only if they are connected in G_p.*

Proof. Since going from UD(T) to G_p only increases the connectivity, all sites s and t connected in UD(T) are also connected in G_p.

For the other direction, let $s = p_1, \ldots, p_k = t$ be a path in G_p between $s, t \in T$. For each p_i, we define

a set $V_i \subseteq T$ as follows: if p_i is a site in T, we set $V_i = \{p_i\}$. Otherwise, p_i is a crossing site, created by a crossing of two edges uv and ab in UD(T). We set $V_i = \{a, b, u, v\}$. By Lemma 8, the sites a, b, u, v are in the same connected component of UD(T). Furthermore, we have $V_{i-1} \cap V_i \neq \emptyset$, since $p_{i-1}p_i$ is a proper subsegment of an edge e in UD(T), and at least one endpoint of e lies in V_{i-1}.

We prove by induction that all sites in $\bigcup_{i=1}^{j} V_i$ lie in the same connected component of UD(T), for $j = 1, \ldots, k$. For $j = 1$, this is clear. Now, consider V_j. If $V_{j-1} \cap V_j \neq \emptyset$, then the claim follows by induction, since all sites in V_j are in the same component. Otherwise, $V_j = \{p_j\}$, p_j is a site in T, and there is an edge in UD(T) between p_j and $\bigcup_{i=1}^{j-1} V_i$, implying the claim. By setting $j = k$, we now have that s and t are connected in UD(T). □

Maintaining G_p. We maintain an MBM between any two neighboring non-empty grid cells and we pick one representative edge for each MBM. Let s be a site we want to insert or delete from S. Let σ be the grid cell containing s. We update for all $\tau \in N(\sigma)$ the MBM $M_{\{\sigma,\tau\}}$, and we collect the sites of all representative edges that need to be inserted or deleted in two sets I and D: if $M_{\{\sigma,\tau\}}$ changes from empty to non-empty, we pick a representative edge for $M_{\{\sigma,\tau\}}$ and put its two endpoints into I. If we delete the representative edge of $M_{\{\sigma,\tau\}}$, we put its two endpoints into D, and, if possible, we pick a new representative edge for $M_{\{\sigma,\tau\}}$. We put the endpoints of the new edge into I. Since $|N(\sigma)| = O(1)$, the sets I and D contain $O(1)$ to be added or deleted from G_p.

Next, we show how to update G_p with a site s in I or D. First we insert or delete s in UD(T) and determine which edges change in UD(T). Each such edge may create or delete several edges in G_p that need to handled. The next lemma shows that s can create or delete $O(1)$ edges in G_p and that these edges can be found in $O(1)$ time. This finishes the proof of Theorem 7.

Lemma 10 *Let s be a site in I or D. Updating G_p with s changes $O(1)$ edges and vertices. They can be found in $O(1)$ time.*

Proof. Suppose that $s \in I$, i.e., we want to insert s. Let σ be the cell containing s. We add s to T and collect all edges in UD(T) incident to s in a set E as follows: we start with $E = \emptyset$. First, for each $t \in T \cap \sigma$ we add the edge st to E. Since σ has diameter 1, all these edges are valid edges in UD(T). Next, we go through all cells $\tau \in N(\sigma)$. We check for all sites $t \in \tau \cap T$ if $|st| \leq 1$. If so, we add st to E.

To update G_p, we find all edges in G_p crossed by edges in E. Since all edges in E and in G_p cross $O(1)$ grid cells, and since each grid cell contains $O(1)$ sites and crossing sites, this can be done in $O(1)$ time. We add all these edges to E, and we perform the planarization procedure on E. This gives all edges and vertices in G_p that need to be changed, in $O(1)$ time.

Deleting a site is done in a similar manner. □

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *DCG*, 6(5):407–422, 1991.

[2] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd FOCS*, pages 617–626, 2002.

[3] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *JACM*, 48(1):1–12, 2001.

[4] T. M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *JACM*, 57(3):Art. 16, 15, 2010.

[5] T. M. Chan, M. Pătraşcu, and L. Roditty. Dynamic connectivity: connecting to networks and geometry. *SICOMP*, 40(2):333–349, 2011.

[6] D. Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *DCG*, 13:111–122, 1995.

[7] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.

[8] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity,minimum spanning tree, 2-edge, and biconnectivity. *JACM*, 48(4):723–760, 2001.

[9] H. Kaplan, R. E. Tarjan, and K. Tsioutsiouliklis. Faster kinetic heaps and their use in broadcast scheduling (extended abstract). In *Proc. 12th SODA*, pages 836–844, 2001.

[10] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *JCSS*, 23(2):166–204, 1981.

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

On Kinetic Range Spaces and their Applications

Jean-Lou De Carufel* Matthew J. Katz† Matias Korman‡ André van Renssen§ ¶
Marcel Roeloffzen§ ¶ Shakhar Smorodinsky‖

Abstract

We study geometric hypergraphs in a kinetic setting and show that for many of the static cases where the VC-dimension of the hypergraph is bounded the kinetic counterpart also has bounded VC-dimension. Among other results we show that for any set of n moving points in $\mathbb{R}^d$ and any parameter $1 < k < n$, one can select a non-empty subset of the points of size $O(k \log k)$ such that each cell of the Voronoi diagram of this subset is "balanced" at any given time (i.e., it contains $O(n/k)$ of the other points). We also show that the bound is near optimal even for the one-dimensional case in which points move linearly.

1 Introduction

Geometric hypergraphs (also called range-spaces) are central objects in computational geometry, statistical learning theory, combinatorial optimization, linear programming, discrepancy theory, data bases and several other areas in mathematics and computer science.

In most of these cases, we have a finite set P of points in $\mathbb{R}^d$ and a family of simple geometric regions, such as the family of all halfspaces in $\mathbb{R}^d$. We then consider the combinatorial structure of the set system $(P, \{h \cap P\})$ where h is any halfspace. Many optimization problems can be formulated on such structures. A key property of such hypergraphs is a bounded VC-dimension (see Section 2 for exact definitions). In this paper we study a more complex structure by allowing the underlying set of points to move along some "reasonable" trajectories. Even though the static case is well-known, little research has been done for the case in which the points move. We show that those more complex hypergraphs defined as the union of all hypergraphs obtained at all possible times still have a bounded VC-dimension. As a result, many deep results that hold for arbitrary hypergraphs with bounded VC-dimension readily apply to such kinetic hypergraphs. By adding several other ingredients, we are able to prove our main result about points moving with bounded description complexity (see below for the exact definitions):

Theorem 1 *Let $P = \{p_1, \ldots, p_n\}$ be any set of n moving points in $\mathbb{R}^d$ with bounded description complexity. For any integer $2 \leq k \leq n$, there exists a subset $N \subset P$ of cardinality $O(k \log k)$, such that for any time $t \geq 0$, each cell of the Voronoi diagram $Vor(N(t))$ contains at most $O(n/k)$ points of $P(t)$.*

The paper is organized as follows: in Section 2 we introduce several key concepts as well as review known results that hold for static range spaces. In Section 3 we extend these results to the kinetic case. In Section 4 we show several applications. Due to lack of space, proofs in this paper are omitted or sketched. Details can be found in the extended version of this paper [2].

2 Preliminaries and Previous Work

We consider the following families of geometric hypergraphs: Let P be a set of points in $\mathbb{R}^d$ and let $\mathcal{R}$ be a family of regions in the same space. We refer to the hypergraph $H = (P, \{P \cap r \colon r \in \mathcal{R}\})$ as the hypergraph induced by P with respect to $\mathcal{R}$. In the literature, such kind of hypergraphs are also referred to as *range spaces*.

Our aim is to show that many properties that hold for static range spaces extend to their kinetic counterparts. We start by introducing some concepts that are frequently used in (static) range spaces.

Recall the definition of an ε-net for a hypergraph: let $H = (V, \mathcal{E})$ be a hypergraph with V finite. Let $\varepsilon \in [0, 1]$ be a real number. A set $N \subset V$ (not necessarily in $\mathcal{E}$) is called an *ε-net* for H if for every hyperedge $S \in \mathcal{E}$ with $|S| \geq \varepsilon |V|$ we have $S \cap N \neq \emptyset$.

A closely related concept of ε-net is the so called Vapnik-Chervonenkis dimension [8]: let $H = (V, \mathcal{E})$ be a hypergraph. A subset $X \subset V$ (not necessarily in $\mathcal{E}$) is said to be *shattered* by H if $\{X \cap S \colon S \in \mathcal{E}\} = 2^X$. The *Vapnik-Chervonenkis dimension*, also

*School of Computer Science, Carleton University, Ottawa, Canada.
†Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel.
‡Tohoku University, Sendai, Japan. Work supported by the ELC project (MEXT KAKENHI No. 24106008).
§National Institute of Informatics (NII), Tokyo, Japan.
¶JST, ERATO, Kawarabayashi Large Graph Project.
‖Department of Mathematics, Ben-Gurion University of the NEGEV, Beer-Sheva, Israel and EPFL, Switzerland. Research partially supported by Swiss National Science Foundation Grants 200020144531 and 200021-137574, and by Grant 1136/12 from the Israel Science Foundation.

denoted the VC-*dimension* of H, is the maximum size of a subset of V shattered by H.

Theorem 2 (ε-net theorem [4]) *Let $H = (V, \mathcal{E})$ be a hypergraph with VC-dimension d. For every $\varepsilon \in (0, 1)$, there exists an ε-net $N \subset V$ with cardinality at most $O(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon})$.*

It is known that whenever range spaces are defined through semi-algebraic sets of constant description complexity (i.e., sets defined as a Boolean combination of a constant number of polynomial equations and inequalities of constant maximum degree), the resulting hypergraph has finite VC-dimension. Halfspaces, balls, boxes, etc. are examples of ranges of this kind; see, e.g., [6, 7] for more details.

Thus, by Theorem 2, these hypergraphs admit "small" size ε-nets. Kómlos *et al.* [5] proved that the bound $O(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon})$ on the size of an ε-net for hypergraphs with VC-dimension d is best possible.

3 Kinetic hypergraphs

In this section we extend the above results to the kinetic model. Let $P = \{p_1, \ldots, p_n\}$ denote a set of n moving points in $\mathbb{R}^d$, where each point is moving along some "simple" trajectory. That is, each p_i is a function $p_i : [0, \infty) \to \mathbb{R}^d$ of the form $p_i(t) = (x_1^i(t), \ldots, x_d^i(t))$. For a given real number $t \geq 0$ and a subset $P' \subset P$, we denote by $P'(t)$ the fixed set of points $\{p(t) \colon p \in P'\}$. For a family of ranges $\mathcal{R}$, we define its induced *kinetic hypergraph* as follows:

Definition 1 (kinetic hypergraph) *Let P be a set of moving points in $\mathbb{R}^d$ and let $\mathcal{R}$ be a family of ranges. Let $(P, \mathcal{E})$ denote the hypergraph where $\mathcal{E}$ consists of all subsets $P' \subseteq P$ for which there exists a time t and a range $r \in \mathcal{R}$ such that $P'(t) = P(t) \cap r$. We call $(P, \mathcal{E})$ the* kinetic hypergraph *induced by $\mathcal{R}$.*

As in the static case we abuse the notation and denote the hypergraph by $(P, \mathcal{R})$. In order to apply our techniques, we need the following "bounded description complexity" assumption concerning the movement of the points of P. We say that a point $p_i = p_i(t) = (x_1^i(t), \ldots, x_d^i(t)) \in P$ moves with *description complexity* $s > 0$ if for each $1 \leq j \leq d$ it holds that $x_j^i(t)$ is a univariate polynomial of degree at most s. In the remainder of this paper, we assume that all points of P move with bounded description complexity, that is, the description complexity s is a constant.

3.1 VC-Dimension of kinetic hypergraphs

In this section we prove that for many of the static range spaces that have small VC-dimension, their kinetic counterparts also have small VC-dimension. We start with the family $\mathcal{H}_d$ of all halfspaces in $\mathbb{R}^d$.

Theorem 3 *Let $P \subset \mathbb{R}^d$ be a set of moving points with bounded description complexity s. Then, the kinetic-range space $(P, \mathcal{H}_d)$ has VC-dimension bounded by $O(d \log d)$.*

To prove Theorem 3, we need the following known definition and lemma (see, e.g., [6]). The *primal shatter function* of a hypergraph $H = (V, \mathcal{E})$ denoted by π_H is a function:

$$\pi_H : \{1, \ldots, |V|\} \to \mathbb{N}$$

defined by $\pi_H(i) = max_{V' \subseteq V, |V'|=i} |H[V']|$, where $|H[V']|$ denotes the number of hyperedges in the sub-hypergraph $H[V']$.

Lemma 4 *Let $H = (V, \mathcal{E})$ be a hypergraph whose primal shatter function π_H satisfies $\pi_H(m) = O(m^c)$ for some constant $c \geq 2$. Then the VC-dimension of H is $O(c \log c)$.*

Proof. [Proof of Theorem 3] By Lemma 4 it suffices to bound the primal shatter function $\pi_{\mathcal{H}_d}(m)$ by a polynomial of constant degree. It is a well known fact that the number of combinatorially distinct half-spaces determined by n (static) points in $\mathbb{R}^d$ is $O(n^d)$. This can be easily seen by charging hyperplanes to d-tuples of points (using rotations and translations) and observing that each tuple can be charged at most a constant (depending on the dimension d) number of times. Thus, at any given time, the number of hyperedges is bounded by $O(n^d)$. Next, note that as t varies, a combinatorial change in the hypergraph $(P(t), \mathcal{R})$ can occur only when $d+1$ points $p_1(t), \ldots, p_{d+1}(t)$ become affinely dependent. Indeed, a hyperedge is defined by a hyperplane that contains d points of $P(t)$, and that hyperedge changes when an additional point of $P(t)$ crosses the hyperplane (and thus $d + 1$ points become affinely dependent). This happens if and only if the following determinant condition holds:

$$\begin{vmatrix} x_1^1(t) & x_2^1(t) & \cdots & x_d^1(t) & 1 \\ x_1^2(t) & x_2^2(t) & \cdots & x_d^2(t) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{d+1}(t) & x_2^{d+1}(t) & \cdots & x_d^{d+1}(t) & 1 \end{vmatrix} = 0 \qquad (1)$$

where $x_i^j(t)$ denotes the i'th coordinate of $p_j(t)$. The left side of the equation is a univariate polynomial of degree at most ds. By our general position assumption this polynomial is not zero for some instant of time. In particular, it cannot be identically zero and thus it can have at most ds solutions.

That is, a tuple of $d + 1$ points of $P(t)$ generates at most ds events. Hence, the total number of such events is bounded by $O(\binom{n}{d+1}) = O(n^{d+1})$. Between any two events we have $O(n^d)$ distinct halfspaces,

thus we can have $O(n^{2d+1})$ distinct ranges distributed along all instants of time.

Since each hyperedge is defined by the points on its boundary, this property is hereditary. That is, for any subset $P' \subseteq P$ the hypergraph $H[P']$ has at most $O(|P'|^{2d+1})$ hyperedges. Thus, the shatter function satisfies $\pi_H(m) = O(m^{2d+1})$. Then by Lemma 4, $(P, \mathcal{H}_d)$ has bounded constant VC-dimension, where the constant depends only on d and s. □

Theorem 3 can be further generalized to arbitrary ranges with so-called bounded description complexity as defined below:

Theorem 5 *Let $\mathcal{R}$ be a collection of semi-algebraic subsets of $\mathbb{R}^d$, each of which can be expressed as a Boolean combination of a constant number of polynomial equations and inequalities of maximum degree c (for some constant c). Let P be a set of moving points in $\mathbb{R}^d$ with bounded description complexity. Then the kinetic range-space $(P, \mathcal{R})$ has bounded* VC*-dimension.*

Proof. The proof combines Lemma 4 with Theorem 3 and the so-called Veronese lifting map from Algebraic Geometry. Proof is similar to the proof for the static case. See, e.g., [6]. □

4 Applications

Balanced Voronoi cells for moving points

Given a set P of moving points or *clients*, we are interested in locating k *facilities* so that at each instant of time no facility serves too many clients (assuming that each client goes to its nearest facility).

Proof. [Sketch of the proof of Theorem 1] First we show that the kinetic hypergraph $H = (P, \mathcal{W})$ induced by all bounded cones (i.e. the intersection of an infinite cone with a ball centered at the apex of the cone) has constant VC-dimension, and construct an ε-net N for H with $\varepsilon = \frac{1}{k}$ (this net is certified to exist thanks to Theorems 2 and 3). We claim that N satisfies the desired property.

Let C_d be the minimum number of sixty degrees caps that are needed to cover the unit sphere $\mathcal{S}^{d-1}$ (such a constant always exists and depends only on d). Assume to the contrary that the Voronoi cell of $q(t)$ contains a subset $P'(t) \subset P(t)$ of more than $C_d n/k$ points. By the pigeonhole principle, at least $n/k + 1$ of the points of $P'(t)$ lie in a 60-degrees infinite cone W and has $q(t)$ as its apex. Consider the (inclusionwise) smallest bounded cone $W' \in \mathcal{W}$ such that $W' \subset W$ and both cones contain the same points of $P(t)$. Translate W' infinitesimally so that $q(t)$ is not in the cone, but no other point enters or leaves the cone. This cone has more than n/k points of $P'(t)$, so it must contain another point $q'(t)$ of $N(t)$. By the triangle inequality, there must be a point in that cone whose closest point is $q'(t)$, contradicting the fact that it belongs to Voronoi cell of $q(t)$. □

Corollary 6 *Let $N \subset P$, $|N| = O(k \log k)$, as in Theorem 1. Then, for any finite point set $S \subset \mathbb{R}^d$, and for any $t \geq 0$, the cell of any $q \in S$ in the Voronoi diagram $Vor(N(t) \cup S)$ contains at most $O(n/k)$ points of $P(t)$.*

Remark We note that the bound of $O(k \log k)$ in Theorem 1 is near optimal already for $d = 1$ and points moving linearly. This follows from a recent lower-bound construction of Alon [1] for ε-nets for static hypergraphs consisting of points with respect to strips in the plane. See more details in [2].

Low interference for moving transmitters

In the following we define the concept of (receiver-based) *interference* of a set of ad-hoc sensors [9]. Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$ and let $r_1, \ldots, r_n$ be n non-negative reals representing the transmission radii assigned to the points $p_1, \ldots, p_n$, respectively. Let $G = (P, E)$ be the graph associated with this power assignment, where $E = \{\{p, q\} : d(p, q) \leq min\{r_p, r_q\}\}$. Let $D = \{d_1, \ldots, d_n\}$ denote the family of balls where d_i is the ball centered at p_i and having radius r_i.

Let $I(D)$ denote the maximum depth of the arrangement of the balls in D. We call $I(D)$ *the interference* of D. Note that both G and $I(D)$ are determined by P and $r_1, \ldots, r_n$. Given a set P of points in $\mathbb{R}^d$, the *interference minimization problem* asks for the power assignment with smallest possible interference among the assignments whose underlying graph is connected.

Empirically, it has been observed that networks with high interference have large probability of messages colliding [9]. Thus, a significant amount of research has focused in the creation of connected networks with low interference. Among other results, we highlight the one of Halldórsson and Tokuyama [3] that any pointset P of n points in $\mathbb{R}^d$ has a radius assignment whose associated interference is $O(\sqrt{n})$.

Here, we turn our attention to the kinetic version of the interference problem in arbitrary but fixed dimension. We wish to maintain a connected graph of a set of moving points that always has low interference. Unless the distances between sensors remain constant, no static radii assignment can work for a long period of time since points will eventually be far from each other. Instead, we describe the network in a combinatorial way. That is, we look for a function $f : P \times [0, \infty) \to P$ that determines, for each sensor of P and instant of time, its furthest away sensor that

must be reached. Then, at time t the communication radius of a sensor $p \in P$ is simply set equal to the distance between p and $f(p,t)$. Ideally, we would like to construct a network that not only has small interference for all instants of time, but also limits the number of combinatorial changes in the graph along time.

Theorem 7 *Let P be a set of n moving points in $\mathbb{R}^d$ with bounded description complexity s. Then there is a power assignment such that at any given time t we have $I(P(t)) = O(\sqrt{n \log n})$. Moreover, the total number of combinatorial changes in the network is at most $O^*(n^{1.5}\sqrt{\log n})$ where the O^* notation hides a term involving the inverse Ackermann function that depends on d and s.*

Proof. [Sketch] As in the static case [3], we use Theorem 1 for $k = \sqrt{n/\log n}$ to obtain a set N of size $O(k \log k)$ with the properties guaranteed by Theorem 1. The elements of N are called *hubs*, and we map to each point of N its furthest point in P. We map non-hub points $p \in P \setminus N$ to their nearest hub. Equivalently, if we consider the Voronoi diagram with sites $N(t)$, function $f(p,t)$ will match point $p(t)$ with the site associated to the Voronoi cell that contains $p(t)$ at time t (the mapping for points of N is similar, but we would use the farthest point Voronoi diagram instead).

Each point of $P \setminus N$ has radius large enough to reach one point of N, and all points of N form a clique, thus the network is connected. Regarding interference, by Corollary 6, we know that no point $q \in \mathbb{R}^d$ can be reached by more than $O(n/k)$ points of $P \setminus N$ at any instant of time. Thus, the total interference of any point $q \in \mathbb{R}^d$ is at most $O(k \log k)$ from hubs (since $|H| = O(k \log k)$), and at most $O(n/k)$ from non-hubs. Since $k = \sqrt{n/\log n}$ we obtain the claimed bound.

The bound on the number of combinatorial changes follows from the fact that we are tracking changes in upper and lower envelopes of a family of one-dimensional functions (i.e., pairwise distances along time). For any two points $p, p' \in P$, the function $d(p(t), p'(t))$ is algebraic of bounded degree. Thus, any two such functions cross $O(s)$ times. This allows us to use the Davenport-Schinzel Theorem to bound the overall number of combinatorial changes of the network. □

5 Conclusions

We showed that for many range-spaces with bounded VC-dimension, the kinetic version of such range-spaces, a more complex and rich structure, still has a bounded VC-dimension. We believe that the boundedness of the VC-dimension of the kinetic hypergraphs is of independent interest and hope that further research will reveal more applications (some of which can be found in [2]).

References

[1] N. Alon. A non-linear lower bound for planar epsilon-nets. *Discrete & Computational Geometry*, 47(2):235–244, 2012.

[2] J. D. Carufel, M. Katz, M. Korman, A. van Renssen, M. Roeloffzen, and S. Smorodinsky. On kinetic range spaces and their applications. *CoRR*, abs/1507.02130, 2015.

[3] M. Halldórsson and T. Tokuyama. Minimizing interference of a wireless ad-hoc network in a plane. *Theoretical Computer Science*, 402(1):29–42, 2008.

[4] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[5] J. Komlós, J. Pach, and G. Woeginger. Almost tight bounds for epsilon-nets. *Discrete & Computational Geometry*, 7:163–173, 1992.

[6] J. Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., 2002.

[7] J. Pach and P. K. Agarwal. *Combinatorial Geometry*. Wiley Interscience, 1995.

[8] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[9] P. von Rickenbach, R. Wattenhofer, and A. Zollinger. Algorithmic models of interference in wireless ad hoc and sensor networks. *IEEE/ACM transactions on sensor networks*, 17(1):172–185, 2009.

Finding the k-Visibility Region of a Point in a Simple Polygon in the Memory-Constrained Model

Yeganeh Bahoo* Bahareh Banyassady† Prosenjit Bose‡ Stephane Durocher*
Wolfgang Mulzer†

Abstract

We study the problem of k-visibility in the memory-constrained model. In this model, the input resides in a randomly accessible read-only memory of $O(n)$ words with $O(\log n)$ bits each. An algorithm can read and write $O(s)$ additional words of workspace during its execution, and it writes its output to write-only memory. In a given polygon P, for a given point $q \in P$, we say a point p is inside the k-visibility region of q iff the segment pq intersects the boundary of P at most k times. Given a simple n-vertex polygon P stored in a read-only array and a point $q \in P$, we give a time-space trade-off algorithm which reports a suitable representation of the k-visibility region of q in $O(n^2/s+n\log s)$ time using $O(s)$ words of workspace.

1 Introduction

Memory constraints on mobile and distributed devices have led to an increasing concern among researchers to design algorithms that use memory efficiently. One common model to capture this notion is the *memory-constrained model* [2]. In this model, the input resides in a randomly accessible read-only array of $O(n)$ words with $O(\log n)$ bits each. There is an additional read/write memory consisting of $O(s)$ words of $O(\log n)$ bits each, called the *workspace* of the algorithm. Here, $s \in \{1, \ldots, s\}$ is a parameter of the model. The output is written to a write-only array.

For a given polygon P and a given point $q \in P$, the point $p \in P$ is *k-visible* from q iff the segment pq properly intersects the boundary of P at most k times (p and q are not counted toward k). The set of k-visible points of P from q is called the *k-visibility region* of q within P, and is denoted $V_k(P, q)$; see Figure 1. Visibility has a rich history in computational geometry and other fields; see [6] for an overview. While the 0-visibility region is a connected component, the k-visibility region may be disconnected. The k-visibility region of a point inside the plane in presence of a polygon can be computed in $O(n^2)$ time [3].

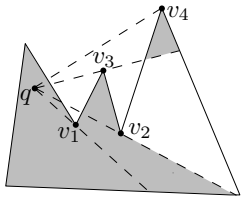


Figure 1: The gray region is $V_2(P, q)$. The vertices v_1, v_2, v_3 and v_4 are critical for q. ∂P is partitioned into chains v_2v_3, v_3v_1, v_1v_4 and v_4v_2.

Using constant workspace, the 0-visibility region of a point $q \in P$ can be computed in $O(n\bar{r})$ time, where $\bar{r}$ denotes the number of the reflex vertices of P in the output [4]. When the workspace is increased to $O(s)$, the running time decreases to $O(nr/2^s + n\log^2 r)$ or $O(nr/2^s + n\log r)$ randomized expected time, where $s \in O(\log r)$. Computing the 0-visibility region without workspace limitations takes $O(n)$ time [1].

We provide time-space trade-off algorithms for computing the k-visibility region of a simple polygon P from $q \in P$ using a small workspace.

2 Preliminaries and definitions

We have a simple polygon P in a read-only array as a list of n vertices in counterclockwise order along the boundary and a query point $q \in P$. The aim is to report a suitable representation of $V_k(P, q)$, using $O(s)$ words of workspace. We assume that the vertices of P are in *weak general position*, i.e., q does not lie on the line determined by any two vertices of P. W.l.o.g., assume that k is even and that $k < n$. If k is odd, we compute $V_{k-1}(P, q) = V_k(P, q)$, and if $k \geq n$, then P is completely k-visible. The boundary of $V_k(P, q)$ consists of part of the boundary of P and some chords that cross the interior of P to join two points on its boundary. We denote the boundary of planar set U by ∂U. Let $\theta \in [0, 2\pi)$, and let r_θ be the ray from q that forms an angle θ with the positive-horizontal axis. The j^{th} edge of P that intersects r_θ, starting from q, is denoted $e_\theta(j)$. Only the first $k+1$ intersections of $r_\theta \cap \partial P$ are k-visible from q in direction θ.

*Department of Computer Science, University of Manitoba, {bahoo, durocher}@cs.umanitoba.ca

†Institut für Informatik, Freie Universität Berlin, {bahareh, mulzer}@inf.fu-berlin.de. Supported by DFG project MU/3501-2

‡School of Computer Science, Carleton University, jit@scs.carleton.ca

If r_θ does not stab any vertices of P, then the *edge lists*, i.e., the list of intersecting edges, of both $r_{\theta-\varepsilon}$ and $r_{\theta+\varepsilon}$, for a small enough $\varepsilon > 0$, are the same as the edge list of r_θ. However, if r_θ stabs a vertex v of P, then the edge lists of $r_{\theta-\varepsilon}$ and of $r_{\theta+\varepsilon}$ differ, for any small $\varepsilon > 0$. The difference is caused by the edges incident to v. If these edges lie on opposite sides of r_θ, then the edge list of $r_{\theta+\varepsilon}$ can be obtained from the edge list of $r_{\theta-\varepsilon}$ by exchanging the name of the corresponding edge. However, if both incident edges of v lie on the same side of r_θ, then there are two edges in the edge list of either $r_{\theta-\varepsilon}$ or $r_{\theta+\varepsilon}$ which are not in the edge list of the other. In this case, we call v a *critical* vertex; see Figure 1. The number of critical vertices in P is denoted by c. The *angle* of a vertex v refers to the angle between the ray qv and positive-horizontal axis. A *chain* is defined as a maximal sequence of edges of P which does not contain a critical vertex, except at the beginning and at the end. Thus, ∂P is partitioned into disjoint chains; see Figure 1.

Observation 1 *Let C be a chain on P. Suppose we are given an edge e of C, and a ray r_θ. We can find the edge $e_\theta \in C$ which intersects r_θ (if it exists) in $O(|C|)$ time using $O(1)$ workspace.*

When rotating the ray r_θ around q, the structure of the edge list of r_θ (i.e., the chains and their order) changes only when r_θ stabs a critical vertex. We will see that in this case a segment of r_θ may belong to $\partial V_k(P,q)$. A critical vertex v on r_θ is counted as both $e_\theta(j)$ and $e_\theta(j+1)$, if there are $j-1$ intersecting edges with r_θ between q and v. Obviously, v is k-visible if its position on r_θ is not after $e_\theta(k+1)$. A critical vertex v is called an *end vertex* if its edges lie on the right side of qv, and it is called a *start vertex* otherwise.

Lemma 1 *If r_θ stabs a k-visible critical vertex v, then the segment on r_θ between $e_\theta(k+2)$ and $e_\theta(k+3)$ (if they exist) is an edge of $V_k(P,q)$.*

Proof. If v is an end vertex, then for small enough $\varepsilon > 0$, the edges $e_\theta(k+2)$ and $e_\theta(k+3)$ are respectively $e_{\theta-\varepsilon}(k+2)$ and $e_{\theta-\varepsilon}(k+3)$, so they are not k-visible in direction $\theta-\varepsilon$. These edges are also $e_{\theta+\varepsilon}(k)$ and $e_{\theta+\varepsilon}(k+1)$, so they are k-visible in direction $\theta+\varepsilon$. Hence, the segment on r_θ between $e_\theta(k+2)$ and $e_\theta(k+3)$ belongs to $\partial V_k(P,q)$, and $V_k(P,q)$ lies on the side of the segment which has direction $\theta+\varepsilon$; see Figure 2. Similarly, if v is a start vertex, the same segment belongs to $\partial V_k(P,q)$; in this case, $V_k(P,q)$ lies on the side of the segment with direction $\theta-\varepsilon$. □

Lemma 1 leads to the following definition: for a ray r_θ that stabs a k-visible critical vertex v, the segment between $e_\theta(k+2)$ and $e_\theta(k+3)$ (if they exist) is called the *window* of r_θ. The window is *CCW* if $V_k(P,q)$ lies to the left of r_θ;(see Figures 2), and *CW*, otherwise.

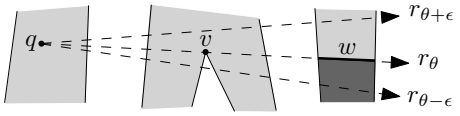


Figure 2: For the ray r_θ which stabs the end vertex v, the segment w is a *CCW* window of $V_4(P,q)$.

Each window is identified by its two endpoints, and each endpoint is represented by a triple (θ, j, type), where j is the index of either $e_\theta(k+2)$ or $e_\theta(k+3)$ in P (depending on the position of two endpoints of a window on these edges) and type $\in \{\text{CCW}, \text{CW}\}$ specifies the *type* of the window. The set of endpoints of windows of $V_k(P,q)$ is denoted by $W_k(P,q)$.

Observation 2 *$\partial V_k(P,q)$ has $O(n)$ vertices.*

Lemma 2 *If there exists an algorithm $A(P,q,k)$ in the memory-constrained model for computing $W = W_k(P,q)$ in $T_A(n)$ time using $S_A(n)$ workspace, where n is the number of vertices of P, then there exists an algorithm $A'(P,q,W)$ in the memory-constrained model that reports $\partial V_k(P,q)$ in $O(|W|T_A(n)+n)$ time using $O(S_A(n))$ workspace.*

Proof. The algorithm A' works as follows: start from a point $w_0 \in W$ and walk on ∂P in CCW direction until the next element $w_1 \in W$. If this walk is on the k-visible side of w_0 (which is specified by the type of w_0), report the visited edges of P; otherwise, report only the windows with endpoint(s) w_0 and/or w_1. Repeat this procedure until the entire boundary ∂P has been traversed. Specifically, in step i of A', run algorithm A and find $w_i = (\theta_i, j_i, \text{type}_i)$ which minimizes j_i, with $j_i > j_{i-1}$ for $i \neq 0$. If there is more than one element which minimizes j_i, choose the one among them that minimizes $|\theta_i - \theta_{i-1}|$ (minimizes θ_i for $i = 0$). Since the output of A is write-only, in each step i of A' we have to run A again to find w_i, requiring $O(|W|T_A(n))$ total time. Regarding the workspace, in step i of A' we store only w_{i-1} and w_i; however, for finding w_i we need as much workspace as A does. Thus, the workspace of A' is $O(S_A(n))$. □

Lemma 2 shows that given $W_k(P,q)$ and P, we can uniquely report $\partial V_k(P,q)$. This motivates us to focus on algorithms for computing $W_k(P,q)$. We assume that P has at least one critical vertex, if not, then $\partial V_k(P,q) = \partial P$. From now on, $e_i(j)$ denotes the j^{th} intersecting edge of the ray qv_i, where v_i is a critical vertex of P. However, instead of $e_i(j)$, it suffices to find an arbitrary edge of the chain containing $e_i(j)$ and then apply Observation 1 to find $e_i(j)$. Therefore, we refer to any edge of the chain containing $e_i(j)$ by $e_i(j)$. The following algorithms, for any critical vertex v_i, examine its position relative to $e_i(k+1)$ on qv_i and,

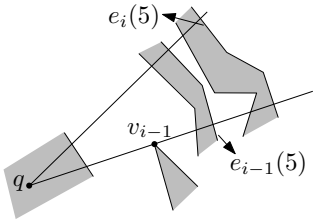


Figure 3: v_{i-1} is an end vertex. $e_i(5)$ is the second intersecting chain to the right of $e_{i-1}(5)$.

if it is k-visible, reports the segment on qv_i which is between $e_i(k+2)$ and $e_i(k+3)$ (if they exist).

3 A constant-memory algorithm

In this section, we assume that only $O(1)$ workspace is available. Suppose that v_0 is the critical vertex with smallest angle. The algorithm starts from qv_0 and finds $e_0(k+1)$ in $O(kn)$ time using $O(1)$ workspace. Basically, the algorithm passes over the input $k+1$ times, and in each pass, it finds the next intersecting edge of qv_0 until the $(k+1)^{\text{th}}$ one, $e_0(k+1)$. If v_0 does not lie after $e_0(k+1)$ on qv_0, in other words, if v_0 is k-visible, it reports the window of qv_0. Finding the window can be done in two passes by determining the first and the second intersecting edge after $e_0(k+1)$ on qv_0. Then, the algorithm finds the next critical vertex with smallest angle after v_0; we call it v_1. The algorithm determines $e_1(k+1)$, and if v_1 is k-visible, it reports the window of qv_1 (if it exists). However, for $1 \leq i$, we find $e_i(k+1)$ in $O(n)$ time by using $e_{i-1}(k+1)$. More precisely, if v_{i-1} is an end vertex, then the incident edges to v_{i-1} do not intersect qv_i; see Figure 3. If v_i is a start vertex, then the incident edges to v_i do not intersect qv_{i-1}. Except for these edges, all the other intersecting edges of qv_{i-1} intersect qv_i in the same order, and vice versa. Hence, if $e_i(k+1)$ intersects qv_{i-1}, then there is at most one other edge between $e_{i-1}(k+1)$ and $e_i(k+1)$ that intersects qv_{i-1}. Thus, $e_i(k+1)$ can be found in at most two passes over the input. More accurately, we have found only an edge of the chain of $e_i(k+1)$; applying Observation 1, the edge $e_i(k+1)$ can be obtained. The algorithm repeats the above procedure until all critical vertices have been processed. Since the number of critical vertices is c, and since processing each critical vertex takes $O(n)$ time, except for v_0, which takes $O(kn)$ time, the running time of the algorithm is $O(kn+cn)$, using $O(1)$ workspace. This leads to the following theorem:

Theorem 3 *Given a simple polygon P with n vertices in a read-only array, a point $q \in P$, and a constant $k \in \mathbb{N}$, there is an algorithm which reports $W_k(P,q)$ in $O(kn+cn)$ time using $O(1)$ workspace.*

4 Memory-constrained algorithms

In this section, we assume $O(s)$ workspace is available, and we show how to exploit this for a faster algorithm. The following lemma is implicitly mentioned in [5] (the second paragraph in the proof of Theorem 2.1)

Lemma 4 *Given a read-only array A of size n, $O(s)$ additional workspace and a specific element $x \in A$, there is an algorithm that finds the s smallest elements in A that are larger than x in $O(n)$ time.*

Proof. In the first step, insert the first $2s$ elements of A that are larger than x into workspace memory (without sorting them). Select the median of the $2s$ elements in memory in $O(s)$ time, and remove the elements which are larger than the median. In the next step, insert the next batch of s elements of A that are larger than x into memory and again find their median. Remove the elements larger than the median. Repeat the latter step until all elements of A are processed. Clearly, at the end of each step, the s smallest elements among those processed so far are in memory. Since the number of batches is $O(n/s)$, the running time is $O(n)$ using $O(s)$ workspace. □

Lemma 5 *Given a read-only array A of size n and $O(s)$ additional workspace, there is an algorithm that finds the k^{th} smallest element in A in $O(\lceil k/s \rceil n)$ time.*

Proof. In the first step, apply Lemma 4 to find the first batch of s smallest elements in A and to insert them into memory in $O(n)$ time. If $k < s$, select the k^{th} smallest element in memory in $O(s)$ time; otherwise, find the largest element in memory. In step i, apply Lemma 4 to find the i^{th} batch of s smallest elements of A and insert them into memory. If $k < i \cdot s$, select the $(k-(i-1)s)^{th}$ smallest element in memory in $O(s)$ time; otherwise, find the largest element in memory and repeat. The element being sought is in the $\lceil k/s \rceil^{th}$ batch of s smallest elements; therefore,the running time is $O(\lceil k/s \rceil n)$ using $O(s)$ workspace. □

There is an $O(n \log\log_s n)$ expected time randomized algorithm for the selection problem using $O(s)$ workspace in the read-only model [7]. Depending on k, s, and n, we choose the latter algorithm or the one in Lemma 5. In each iteration of the following algorithm, we find the next batch of s critical vertices with smallest angles and sort them in memory. Then, we construct a data structure T that contains the possible candidates for the $(k+1)^{\text{th}}$ intersecting edges of the rays from q to the critical vertices of the batch. In each step, when we process a critical vertex of the batch, we use T to find the window of the critical vertex, and we update T. For updating T efficiently, we use another data structure T_θ; see below. We repeat this procedure for the next batch of s critical vertices.

As in the constant-memory algorithm, we find the critical vertex v_0 with smallest angle. We apply Lemma 4 to find the batch of s critical vertices with smallest angles after v_0, and we sort them in memory. For qv_0, we apply Lemma 5 to find $e_0(k+1)$, and if v_0 is k-visible, we report the window (if it exists). Then, we apply Lemma 4 to find the two batches of $2s$ adjacent intersecting edges to the right and to the left of $e_0(k+1)$ on qv_0, we insert them in a balanced search tree T. Hence, T stores all $e_0(j)$, for $k+1-2s \leq j \leq k+1+2s$, in sorted order according to their intersection with qv_0. These edges are candidates for the $(k+1)^{\text{th}}$ intersecting edge of the next s rays in angular order or $e_i(k+1)$, for $1 \leq i \leq s$. This is because, as we explained before, if $e_i(k+1)$ intersects qv_{i-1}, then there is at most one other edge between $e_{i-1}(k+1)$ and $e_i(k+1)$ that intersects qv_{i-1}. Therefore, $e_i(k+1)$ is either an intersecting edge of qv_0, and in this case there are at most $2i-1$ edges between $e_0(k+1)$ and $e_i(k+1)$, or $e_i(k+1)$ is an edge which is inserted in T later. Then for each edge in T we determine the larger angle of its endpoints. This angle shows the position of the endpoint between the rays from q to the critical vertices. Specifically, if the edge is incident to a non-critical vertex, this angle determines the step in which the name of the edge in T should be updated to the other incident edge to the vertex. By traversing ∂P we determine these angles for the edges in T, and we insert them in a balanced search tree T_θ, whose entries are connected through cross-pointers to their corresponding edges in T. We construct T_θ in $O(n + s\log s)$ time.

After creating T and T_θ, we start from the next critical vertex with smallest angle after v_0, called v_1, and we update T so that it contains the edge list of qv_1: If there is any angle in T_θ which is smaller than the angle of v_1, we change the corresponding edge of the angle in T with its previous or next edge in P. In other words, we have found a non-critical vertex between qv_0 and qv_1 and so we change its incident edge, which has been already in T, with its other incident edge. Then we find the angle of the new edge and insert it into T_θ. These two steps take $O(1)$ and $O(\log s)$ time for each angle that meets the condition. By doing these steps, changes in the edge list which are caused by non-critical vertices between qv_0 and qv_1 are handled. Then we update T and consequently T_θ according to the type of v_1: if v_1 is an end (start) critical vertex, we remove (insert) the two edges which are incident to v_1. In both cases, we update T only if the incident edges to v_1 are in the interval of the $2s$ intersecting edges of qv_0 in T, this takes $O(\log s)$ time. Now T contains $2s$ intersecting edges of qv_1, and we can find $e_1(k+1)$ using the position of $e_0(k+1)$ and its neighbours in T in $O(1)$ time. We repeat this procedure for $1 \leq i \leq s$, and we determine $e_i(k+1)$ and the window of qv_i by using T and $e_{i-1}(k+1)$.

After processing the first batch, we apply Lemma 4 to find the next batch of s critical vertices with smallest angle, and we sort them in memory. The last updated T is not usable anymore, because it does not necessarily contain any right or left neighbours of $e_s(k+1)$. Applying Lemma 4 as before, we find the two batches of $2s$ adjacent intersecting edges to the right and to the left of $e_s(k+1)$ on qv_s and we insert them into T. We also update T_θ. Then similarly for each $s < i \leq 2s$, we find $e_i(k+1)$ and its corresponding window, and we update T and T_θ. In summary, updating T considering the changes that are caused by critical and non-critical vertices of the batch takes respectively $O(s\log s)$ and $O(n'\log s)$ time, where n' is the number of non-critical vertices that lie on the interval of the batch. In the next iteration, we repeat the same procedure for the next batch of critical vertices. We stop when all critical vertices are processed. Since the batches do not have any intersections, each non-critical vertex lies only on one batch. Thus, updating T in all batches takes $O(n\log s)$ time. All together, finding the batches of s critical vertices, constructing and updating the data structures and reporting the windows take $O(cn/s+n\log s)$ time for all the critical vertices, in addition to the running time of k-selection in the first batch.

Theorem 6 *Given a simple polygon P with n vertices in a read-only array, a point $q \in P$, and a constant $k \in \mathbb{N}$, there is an algorithm which reports $W_k(P,q)$ in $O(cn/s+n\log s+\min\{kn/s, n\log\log_s n\})$ time using $O(s)$ workspace.*

References

[1] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1-4):49–63, 1986.

[2] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *CGTA*, 46(8): 959–969, 2013.

[3] A. L. Bajuelos, S. Canales, G. Hernández-Peñalver, and A. M. Martins. A hybrid metaheuristic strategy for covering with wireless devices. *J. UCS*, 18(14):1906–1932, 2012.

[4] L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing a visibility polygon using few variables. *JoCG*, 47(9):918–926, 2014.

[5] T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *DCG*, 37(1):79–102, 2007.

[6] S. K. Ghosh. Visibility algorithms in the plane. *Cambridge University Press*, 2007.

[7] J. I. Munro and V. Raman. Selection from read-only memory and sorting with optimum data movement. *Theoretical Computer Science*, 165(2):311-323, 1996.

Minimal Witness Sets For Art Gallery Problems

Eyüp Serdar Ayaz* Alper Üngör*

Abstract

We study the problem of finding witness sets for polygons which can be used as a first step to solve the problem of guarding art galleries. For a polygon P, a set $W \subseteq P$ is called a *witness* set if every set G that guards W, is guaranteed to guard P. Previous study exists for computing a minimal witness set of points for polygons. However, very few polygons admit witness sets of points. Here we propose an algorithm for computing witness sets of points, line segments and, if necessary, regions in $O(n^4)$ time. The output witness set is shown to be minimal if the input polygon has one, otherwise is shown to be near-minimal (as defined later in the paper). This algorithm also determines whether guarding the boundary of a polygon is sufficient to guard the entire polygon.

1 Introduction

The Art Gallery Problem (AGP) was proposed by Klee to Chvatal in 1973 as a challenge to find the point locations of a minimum number of guards such that each point on the walls of an art gallery is seen by at least one guard [3]. In general, gallery interior needs to be guarded as well. Both the original [4] and the generic [6] versions of AGP are proven to be NP-hard. It is clear that a solution for the generic version also guards the wall, but the reverse proposition is not the case. Over the years many other versions of AGP have also been studied as surveyed in [7, 9].

A common approach to solve AGP for a polygon P employs integer programming and uses a formulation with two parameters: AGP(X, Y), where $X, Y \subseteq P$, X is the set of possible guard locations and Y is the set of points to be guarded [8]. Note that AGP(X, P) and AGP(P, Y) are upper and lower bounds respectively on the minimum number of guards. Various heuristics are used to initialize X and Y and then iteratively insert elements into them until lower and upper bounds converge.

The original witnessability problem formulation is credited to Joseph Mitchell in a paper by Chwa *et al.* [2]. A *witness set* W of a polygon P is defined as a set such that if any set G that guards W also guards P. The straight-forward use of witnessability concept is checking the visibility of a subset of a polygon instead of the whole polygon from a guard set. If a witness set consists of lower dimensional elements there can be further algorithmic advantages. Another use is within the initialization step of the integer programming approach discussed above. Using appropriate witness points as Y results in early convergence. Unfortunately, not all polygons admit a finite witness set of points (See Figure 1).

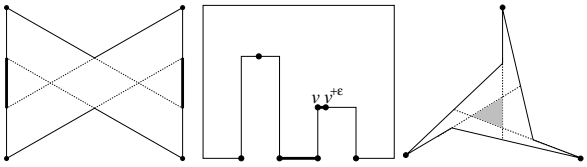


Figure 1: Three polygons admitting no witness set of points only. (Left) polygon has a minimal witness set of points and segments. (Middle) polygon has no minimal witness set but a near-minimal one, as it is necessary to include $\overline{vv^{+\epsilon}}$. (Right) polygon has a minimal witness set of three points and an interior region.

Here we extend the concept of witnessability using sets with points, line segments and if necessary regions. We propose an algorithm that finds a (near)-minimal witness set and, as a consequence, determines whether a solution for AGP(P, ∂P) guarantees a solution for AGP(P, P) i.e., ∂P is a witness set for P.

2 Preliminaries

The input for the witnessability problem is a simple (non-convex) polygon P with n vertices where $int(P)$, and ∂P denote the interior and the boundary of P, respectively. For a reflex vertex v of P, let $v^{-\epsilon}$, $v^{+\epsilon} \in \partial P$ be two infinitesimally close points to v on clockwise and counter-clockwise traversals, respectively. Next, we review some concepts and results from [2].

Two points $p, q \in P$ *see* each other if the whole line segment $\overline{pq}$ is in P. If a point p in P sees a reflex vertex v of P and the ray $\overrightarrow{pv}$ continues in P after hitting v then we say p *sees past* v. If the exterior part of the polygon is on the left side of $\overrightarrow{pv}$ in the immediate neighborhood of v then we say p *sees past left* v. Similarly, if the exterior is on the right, p *sees past right* v. (See Figure 2). For an edge e of P in counter-clockwise orientation, the half-plane induced by e is the set of points on the left side of the line

*CISE Department, University of Florida, Gainesville [ayaz,ungor]@cise.ufl.edu

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

through e and is denoted as $l^+(e)$. For two points $p, v \in P$ such that p sees past left (right) v, let $l^+(p, v)$ denote the half-plane that is right (left) of the line $\overrightarrow{pv}$. The closures of the half-planes $l^+(e)$ and $l^+(p, v)$ are denoted as $l^c(e)$ and $l^c(p, v)$.

The set of points in P that can be seen from a point $p \in P$ is called the *visibility polygon* of p [5], denoted as $\mathcal{V}(p)$. The set of points that can see every point in $\mathcal{V}(p)$ is called the *visibility kernel* of p, denoted as $\mathcal{VK}(p)$. For a set of points S, we use $\mathcal{VK}(S)$ as the union of the visibility kernels of the points in S.

A *witness set* can also be defined as a finite set $W \subseteq P$ such that, for any set of point guards G in P, $W \subseteq \bigcup_{g \in G} \mathcal{V}(g)$ implies $\bigcup_{g \in G} \mathcal{V}(g) = P$. If W is a witness set for P, then W is said to *witness* P. The same verb, to *witness*, can also be used with other geometric entities like points, if seeing the subject guarantees the object to be seen. A witness set W for P is said to be *minimal* if there exists no proper subset of W that witnesses P. If there exists a minimal witness set for P, then P is a *minimalizable* polygon. Otherwise, P is a *non-minimalizable* polygon.

Theorem 1 [2]. *A point set W is a witness set for a polygon P if and only if $\mathcal{VK}(W) = P$. Also the following statements are equivalent for $p, q \in P$: (i) p witnesses q; (ii) $q \in \mathcal{VK}(p)$; (iii) $\mathcal{VK}(q) \subseteq \mathcal{VK}(p)$.*

Let p be a point on the boundary of P. Let $E(p)$ denotes the set of edges of P of which p sees at least one interior point. We define $RM(p)$, as the short form of rightmost vertex of p, the first vertex that p sees past left in counter-clockwise order from the viewpoint of p as the *rightmost* vertex with respect to p and denote as. The $leftmost$ vertex of p, $LM(p)$, is defined symmetrically. If p does not see past left (right) any vertex, then we set $RM(p)(LM(p))$ as the next vertex on ∂P in (counter-)clockwise order. Then, as proven by Chwa *et al.* [2], we have:

$$\mathcal{VK}(p) = l^c(p, RM(p)) \cap l^c(p, LM(p)) \cap \bigcap_{f \in E(p)} l^c(f) \quad (1)$$

3 Minimal and near-minimal witness sets

In this section, we define the lemmas and theorems to be used in our algorithm that finds a (near)-minimal witness set for a simple polygon.

3.1 Witnesses on the boundary of the polygon

We define *anchor points* to subdivide the boundary of the input polygon. Anchor points consist of three types: 1. The vertices of the polygon. 2. For each reflex vertex, the boundary points where the extension of each of the two edges incident to it hit first. 3. For every pair of reflex vertices that see past each other, the boundary points where the two extensions of the line segment between them hit first (See Figure 2). The line segments between two consecutive anchor points are *anchor edges*.

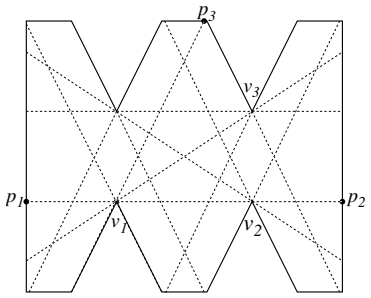


Figure 2: Partition of a polygon boundary and inducing line segments and their extensions. v_1(Type 1), p_3(Type 2) and p_1(Type 3) are three of the 32 anchor points. The line segment $\overline{v_1v_3}$ is a cross line but $\overline{v_1v_2}$ is not. Here, p_1 sees past right v_1 while p_2 sees past left v_1.

Observation 1 *Every point on an anchor edge sees past left and right the same set of reflex vertices (due to Type 2 anchor points). Also these points (partially) sees the same set of edges (due to Type 3 anchor points). Moreover the leftmost and the rightmost reflex vertices a point sees past is the same for all points within an anchor edge.*

We classify the boundary points of P exclusively into five types according to the vertices they see past left and/or right. If a boundary point p doesn't see past any vertex except the ones incident to the edge(s) p belongs to, it is of *Type Z* (See Figure 3). If there exists a vertex p sees past left but there are no vertices p sees past right, then p is of *Type L*. The symmetric version of Type L is *Type R*. If there exist a vertex p sees past left and another vertex p sees past right, there are two possibilities: If $\mathcal{VK}(p) = \{p\}$, then we say that p is of *Type D*. Otherwise, p is of *Type N*. Since the type of points within an anchor edge is the same, we also use these types for anchor edges.

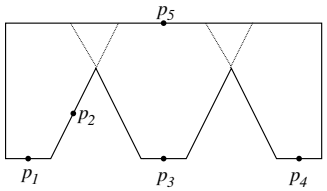


Figure 3: p_1 is of Type R, p_2 is of Type Z, p_3 is of Type N, p_4 is of Type L, p_5 is of Type D

Lemma 2 *Let e be an anchor edge, and p and p' be any two interior points of e in counter-clockwise order.*

If e is of Type L, then p witnesses p'. Similarly if e is of Type R, then p' witnesses p.

Proof. Consider the case e is of Type L. Notice that $E(p) = E(p')$ from Observation 1. Hence, $p' \in \bigcap_{f\in E(p)} l_c(f) = \bigcap_{f\in E(p')} l_c(f)$. Due to the orientation of p and p' along the boundary $p' \in l^c(p, RM(p))$. From (1), we have $p' \in \mathcal{VK}(p)$ and from Theorem 1 p witnesses p'. The symmetric case can be proven similarly. □

Theorem 3 *Let e be an anchor edge, v_1 and v_2 be the endpoints of e in counter-clockwise order, and p be an interior point of e. Let $A = \bigcap_{f\in E(p)} l^c(f)$, $B = l^+(v_1, RM(p)) \cup RM(p)$, and $C = l^+(v_2, LM(p)) \cup LM(p)$. $\mathcal{VK}(e)$ can be calculated using finitely many half-plane intersections:*
(a) If e is of Type D, then $\mathcal{VK}(e) = e$
(b) If e is of Type Z, then $\mathcal{VK}(e) = A$
(c) If e is of Type L, then $\mathcal{VK}(e) = A \cap B$
(d) If e is of Type R, then $\mathcal{VK}(e) = A \cap C$
(e) If e is of Type N, then $\mathcal{VK}(e) = A \cap B \cap C$

Proof. (a) It follows from the definition of Type D.
(b) From (1) we have:

$$\mathcal{VK}(e) = \bigcup_{q\in e} \mathcal{VK}(q) = \bigcup_{q\in e} \bigcap_{f\in E(q)} l^c(f) = A$$

(c) From (1), Theorem 1 and Lemma 2, we have:

$$\begin{aligned}\mathcal{VK}(e) &= \bigcup_{q\in e} \mathcal{VK}(q) = \bigcup_{q\in e} (l^c(q, RM(q)) \cap \bigcap_{f\in E(q)} l^c(f)) \\ &= A \cap \bigcup_{q\in e} l^c(q, RM(q)) = A \cap B\end{aligned}$$

(d) The symmetric case of part (c).
(e) From (1), we know that $\mathcal{VK}(p) \subset A$, hence $\mathcal{VK}(e) \subseteq A$. Also, due to the orientation of the points on e, $A \cap l^c(p, RM(p)) \subset A \cap B$. With this and the symmetrical equivalent, we can see that $\mathcal{VK}(e) \subseteq A \cap B \cap C$.

To prove the equivalence, we need to show that every point in $A \cap B \cap C$ needs to be witnessed by a point on e. When p approaches to v_1, $\mathcal{VK}(p)$ converges to $A \cap B \cap l^c(v_1, LM(p))$. With this and the symmetric version, we have $(A \cap ((B \cap l^c(v_1, LM(p)) \cup (C \cap l^c(v_2, RM(p))) \subset \mathcal{VK}(e)$. For the rest of the points, let r be a point in $(A \cap B \cap C) \setminus (A \cap ((B \cap l^c(v_1, LM(p)) \cup (C \cap l^c(v_2, RM(p)))$. Let r' be the first point the ray $\overrightarrow{v_1 r}$ hit on ∂P. Observing that $r' \in e$ implies $r \in \mathcal{VK}(r')$. □

3.2 Witnesses in the interior of the polygon

For every pair of reflex vertices that see past left each other or see past right each other, line segment between them is called a *cross line* (See Figure 2). A point $p \in P \setminus \mathcal{VK}(\partial P)$ is called a *ordinary point* if it is not on a cross line.

Proofs of Lemmas 4-8 are omitted due to space limitations. They can be found in the complete version.

Lemma 4 *Let p is a point on the cross line $\overline{vu}$ that is not in $\mathcal{VK}(\partial P)$. If p is not on the closure of $\mathcal{VK}(\partial P)$, then $\mathcal{VK}(p) = \{p\}$. Otherwise $\mathcal{VK}(p) \subseteq \overline{vu}$.*

Lemma 5 *A ordinary point p can only witness itself, i.e., $\mathcal{VK}(p) = \{p\}$. Moreover p is present in every witness set of P.*

3.3 Minimal witness sets

Following lemmas establish the groundwork of our results on minimal witness sets.

Lemma 6 *Let p and q be distinct points. If $\mathcal{VK}(p) \subset \mathcal{VK}(q)$, then p cannot be in any minimal witness set.*

Lemma 7 *If a point p is in $\mathcal{VK}(\partial P) \setminus \partial P$ then p cannot be in any minimal witness set.*

Lemma 8 *If a point p on ∂P is not witnessed by another point on ∂P, then p has to be in every minimal witness set.*

Theorem 9 *A minimal witness set consist of a set of boundary elements, all ordinary points, and at least one point on each cross line that is in $P \setminus \mathcal{VK}(\partial P)$.*

Proof. Let $p \in P$. Then p has to be one of these three disjoint sets: $\mathcal{VK}(\partial P)$, ordinary points and points on cross lines that are not in $\mathcal{VK}(\partial P)$. Lemma 5 states that ordinary points are in any witness set including a minimal one. Lemma 7 states that the points that are in $\mathcal{VK}(\partial P) \setminus \partial P$ cannot be in any minimal witness set. Points on cross lines that are not in $\mathcal{VK}(\partial P)$ needs to have at least one point in a minimal witness set according to Lemma 4. □

It is clear that not all polygons admit unique minimal witness sets as we can choose any point on a Type Z anchor edge and, for some cases, any point on a cross line. Other than those, the ordinary points are proven to be necessary in any witness set via Lemma 5. Also the points that are witnessed by ∂P proven not to be in any minimal witness set via Lemma 7. Therefore miminal witness sets for P differ by finitely many points and the cardinalities of them are equal.

3.4 Near-minimal witness sets

Suppose we have an edge segment with points Type L (R) and the left (right) endpoint of the segment is a reflex vertex. Using Lemma 2 we can show that for any point p, there is another point on the left (right)

of p that witnesses p. However the line segment is open and we cannot reach on the left (right) endpoint of the line segment but we need to keep at least one point arbitrarily close to the reflex vertex. For these cases, we use the infinitesimally short line segments $\overline{pp^{-\epsilon}}$ or $\overline{pp^{+\epsilon}}$ in the witness set. (See Figure 1)

Definition 1 *(Near-minimality) Let W be a witness set for a non-minimalizable polygon P. W is* near-minimal *if it can be divided into two disjoint sets, W_{min} and W_ϵ such that W_ϵ consist of finitely many infinitesimally short line segment and removal of any element from W_{min} or W_ϵ makes W to violate the witnessing condition of W. We also call each element of W_ϵ as an* ϵ-witness.

Analogous results to Lemmas 7, 8, and Theorem 9 for near-minimal witness sets are rather immediate, however omitted here for the sake of brevity.

4 Algorithm

We use a subdivision of the polygon based on an arrangement of line segments of following three types: 1. Extensions of edges that are incident to reflex vertices until they hit other boundary points. 2. For each anchor point p, the line segments from p to $LM(p)$ and $RM(p)$. 3. For each anchor edge $\overline{v_1v_2}$, let p be an interior point of $\overline{v_1v_2}$; the line segments from v_1 to $RM(p)$ and from v_2 to $LM(p)$. We denote this subdivision as $\mathcal{A}(P)$ (stored as a doubly connected edge list) and the contiguous regions in the interior of the arrangement as *cells*.

Let W be a candidate (near-)minimal witness set initialized to be empty. For each cell c, we record the number of elements in W that witnesses c as $count(c)$. For each directed line segment s of $\mathcal{A}(P)$, incident to cells c_l and c_r, $\Delta count(s)$ stores $count(c_l) - count(c_r)$. Each line segment is marked once it is witnessed. The algorithm consists of five steps:

1. Find the anchor points and the edges they (partially) see: For this purpose, we simply employ a standard linear time visibility algorithm [5] per vertex resulting in a total cost of $O(n^2)$ time.

2. Compute $\mathcal{A}(P)$: There can be at most $O(n^2)$ line segments, therefore it takes $O(n^2 \log n + k)$ time where k is the number of intersecting points [1], which is in $O(n^4)$. Hence, the total time for this step is $O(n^4)$.

3. Find the elements of a (near-)minimal witness set on ∂P: The visibility kernel of each anchor point or edge is contiguous and inclusive. Therefore we can traverse the boundary of the visibility kernel starting from the anchor point or edge. When we insert an element b to W, we traverse the boundary of $\mathcal{VK}(b)$ in counter-clockwise order and we increment the $\Delta count$ value of each line segment by one and decrement the $\Delta count$ of the opposite direction. When we remove an element from W, which happens when another boundary point witnesses an element on W, we backtrack this increment/decrement. For Type N and D, the whole line segments are inserted to W. To keep the minimality of W, for Type D, we chose the middle point to be inserted to W. For Type L (R), we insert the left (right) endpoint of the line segment to W if it is not a reflex vertex. If the endpoint is a reflex vertex, we insert an ϵ-witness incident to the corresponding end point. Note that the visibility kernels of both anchor points and anchor edges are convex. Therefore, each of $O(n^2)$ line segments of $\mathcal{A}(P)$ can intersect a visibility kernel twice. There can be at most $O(n^2)$ visibility kernels. Therefore this step costs $O(n^4)$ time.

4. Find the cells of a (near-)minimal witness set in $int(P)$: The $count$ values can be retrieved starting from a boundary cell and using the $\Delta count$ values of incident line segments of $\mathcal{A}(P)$. At the end, we insert the cells that have witness $count$ 0 and the unwitnessed line segments of $\mathcal{A}(P)$ that are not in the closure of $\mathcal{VK}(\partial P)$ to W. As a last step, we traverse each line segment s on the boundary of $\mathcal{VK}(\partial P)$. If $s \notin \mathcal{VK}(\partial P)$ we insert either s or a single point of s depending if s is on a cross line. This step is done in $O(n^4)$ time.

If there exists no ϵ-witness element in W then W is minimal. Otherwise the polygon is not minimalizable and W is near-minimal. If step 4 does not find an element in $int(P)$, then ∂P witnesses P. As we can see from the steps, the algorithm works in $O(n^4)$ time.

References

[1] B. Chazelle, H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, *Proc. 29th IEEE Symp. on Found. of Comp. Sci.*, 217-225, 1988.

[2] K.-Y. Chwa, B.-C. Jo, C. Knauer, E. Moet, R. van Oostrum, and C.-S. Shin, Guarding Art Galleries by Guarding Witnesses. *Int. J. Comp. Geometry Appl.*, 16(2-3): 205-226, 2006.

[3] R. Honsberger, *Mathematical Gems II*, Mathematical Association of America, 1976.

[4] A. Laurentini. Guarding the walls of an art gallery, *The Visual Computer*, 15(6):265-278, 1999.

[5] D.T. Lee, Visibility of a simple polygon, *Comp. Vision, Graphics, and Image Processing* 207-221, 1983.

[6] D.T. Lee and A. K. Lin, Computational complexity of art gallery problems. *IEEE Trans. Inf. Theor.*, 32(2):276-282, 1986.

[7] J. O'Rourke, Art Gallery Theorems and Algorithms. Oxford University Press, 1987.

[8] P. J. de Rezende, C. C. de Souza, S. Friedrichs,M. Hemmer, A. Krller and D. C. Tozoni, Engineering Art Galleries. arXiv preprint arXiv:1410.8720.

[9] J. Urrutia, Art gallery and illumination problems. *Handbook on Comp. Geo.*, North-Holland. 973-1027, 2000.

Reconstructing a Unit-Length Orthogonally Convex Polygon from its Visibility Graph

Nodari Sitchinava* Darren Strash†

Abstract

Reconstructing a polygon from its visibility graph is a fundamental problem, and it is still unknown if simple polygons can be reconstructed in polynomial time. We show that a restricted class of polygons—orthogonally convex polygons with unit-length edges—can be reconstructed from an n-vertex visibility graph in $O(n^4)$ time.

1 Introduction

Recognizing visibility graphs and reconstructing their polygons are fundamental problems that are known to be in PSPACE [3], but it is still unknown if simple polygons can be recognized in polynomial time. Therefore, most research has focused on finding efficient algorithms for restricted classes of polygons, such as spiral [4] and funnel polygons [2]. Surprisingly, very few results exist for even orthogonal polygons: we are only aware of efficient algorithms to recognize convex fans, which consist of a single staircase with an additional vertex [1]. Other algorithms for orthogonal polygons assume extra visibility information is given, such as edge-edge visibility [7, Section 7.3], or "vertical stabs," which capture visibility between vertical edges [6]. See Ghosh's book [5] for a thorough review of results on visibility graphs.

2 Preliminaries

Let P be a polygon on n vertices and edges. We say that two points p and q are *visible* in P if line segment pq is in P. Further, a *visibility graph* $G_P = (V_P, E_P)$ of polygon P has a vertex $v_p \in V_P$ for each vertex p of P, and an edge $(v_p, v_q) \in E_P$ when vertices p and q are visible in P. Edges in G_P that are edges of P are called *boundary edges*. Finally, we note that a maximal clique in G_P corresponds to a maximal convex region whose vertices are a subset of P's vertices.

Unit-Length Orthogonal Polygons. Let P be an orthogonal polygon with unit-length edges, such that no three consecutive vertices on P's boundary are collinear. We call P a *unit-length orthogonal polygon*. We call boundary edges between two convex vertices in a unit-length orthogonal polygon P *tab edges* and their vertices are called *tab vertices*.

We first note that there is a simple algorithm to reconstruct a unit-length orthogonal polygon from its visibility graph if we are given the boundary edges.

Observation 1 *Given a visibility graph $G_P = (V_P, E_P)$ with $n = |V_P|$ vertices and $m = |E_P|$ edges of a unit-length orthogonal polygon P and a Hamiltonian cycle $H = v_0, v_1, \ldots, v_{n-1}$ of the boundary edges of P, we can reconstruct P in $O(n + m)$ time.*

Proof. Omitted. □

In this paper, we consider only unit-length polygons that are *orthogonally convex*. That is, any two points in P are visible via a staircase. We call such polygons unit-length orthogonally convex polygons (UPs). We now focus on finding the boundary edges of a UP, which we can use to reconstruct it by Observation 1.

Properties of UPs. We assume that a UP is axis-aligned, allowing us to use the compass analogy. Note that UPs have four tab edges. We call the tab edge with the largest y-coordinate the north edge, and we similarly name the others the south, east, and west edges. Furthermore, the remaining boundary edges are divided into four staircases, which we refer to as northwest, northeast, southeast, and southwest (i.e., staircases do not contain tab edges).

Note that, for brevity, we only consider polygons with more than 12 vertices. This way, we avoid many special cases in the smaller polygons.

Observation 2 *In a visibility graph of a UP, there is exactly one maximal clique containing all reflex vertices. Moreover, this clique contains no tab vertex.*

Lemma 1 *Every convex vertex u in UP, has a convex neighbor v such that (u, v) is in exactly one maximal clique in the visibility graph of the UP.*

Proof. If u is a tab vertex, then the other tab vertex v is also convex and (u, v) is in exactly one maximal clique. Otherwise, suppose w.l.o.g. that u is on the northwest staircase. Then u has a convex neighbor v on the southeast staircase, and (u, v) is in one maximal clique, which consists of u, v, the reflex vertices within the rectangle R defined by u and v as the opposite corners, and any other corners of R that are convex vertices of the polygon. □

*Department of Information and Computer Sciences, University of Hawaii at Manoa, USA. E-mail: nodari@hawaii.edu

†Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany. E-mail: strash@kit.edu

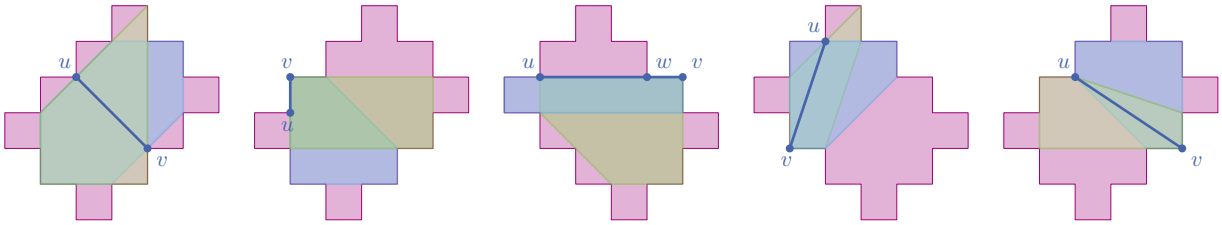


Figure 1: Edges incident to a reflex vertex are contained in at least two maximal cliques.

Lemma 2 *For any edge (u, v) in the visibility graph of a UP, if u or v is reflex, then (u, v) is in at least two maximal cliques. (See Figure 1.)*

Proof. Let $(u, v) \in E_P$ and suppose that at least one of u and v is reflex. Suppose w.l.o.g that u is a reflex vertex on the northwest staircase.

Case 1: Vertex v is reflex. Then u and v are in the maximal clique containing all reflex vertices and no tab vertex, plus u and v see a common tab vertex; therefore (u, v) is in at least two maximal cliques.

For the remaining cases, we assume that v is convex.

Case 2a: Edge (u, v) is a boundary edge. Assume w.l.o.g. that (u, v) is a vertical edge on the northwest staircase and v is above u. If (u, v) is adjacent to the north tab edge, then it is in two maximal cliques: a rectangular clique containing the edge opposite to the tab, and a clique containing an east tab vertex. Suppose (u, v) is not adjacent to the north tab edge. Then u and v see at least 2 convex vertices on the southeast staircase. Thus there are at least two maximal cliques, each containing one of these convex vertices.

We now assume that (u, v) is not a boundary edge.

Case 2b: Edge (u, v) is an axis-aligned visibility edge. We assume w.l.o.g. that (u, v) is a horizontal edge and overlaps boundary edge (v, w). Then v is either on the north- or southeast staircase. If v is on the northeast staircase, u and v are in a maximal clique containing u's boundary neighbor to the west, and one containing u, w, v and all reflex vertex vertices south of w and west of v. If v is on the southeast staircase, then u and v are in a maximal clique containing u's convex boundary neighbor to the north, and another maximal clique containing u, w, v and all reflex vertices north of w and west of v.

Case 2c: Edge (u, v) is a diagonal edge between two adjacent staircases. Then assume w.l.o.g. that v is on the southwest staircase. Then both u and v see a tab vertex t on the north tab. There is a maximal clique containing u, v, t and a maximal clique containing u, v, the reflex vertices north and east of v, and not t.

Case 2d: Edge (u, v) is a diagonal edge between two opposite staircases. Then there is a maximal clique containing u, v, and u's convex boundary neighbor to the west, and another maximal clique containing u, v and u's convex boundary neighbor to the north. □

Therefore we can compute all convex vertices, leading to the following lemma.

Lemma 3 *We can identify all convex and reflex vertices in a visibility graph of a UP in $O(n^4)$ time.*

Proof. For each edge, compute if it is in exactly one maximal clique in $O(n^2)$ time. If so, its endvertices are convex. The remaining vertices are reflex. Checking all $O(n^2)$ edges takes $O(n^4)$ time in total. □

Definition 1 (regularity) *We call a UP* regular *if each of its staircase boundaries have the same number of vertices. Otherwise, we call it* irregular.

For this extended abstract, we concentrate on irregular unit-length orthogonally convex polygons (IUPs). However, similar methods work for regular polygons.

3 Reconstructing IUPs

From now on, we assume that P is an IUP, and that G_P is its visibility graph. We note that two staircases in an IUP have more vertices than the other two. We call these *long* staircases, and the other ones *short*.

Lemma 4 *An IUP has 4 maximal cliques of size 7 that contain more than 2 convex vertices. Furthermore, each such clique contains exactly one tab edge.*

Proof. Each of the 4 tabs are in exactly one such maximal clique. Other cliques that contain 3 convex vertices have at least 9 vertices, each convex vertex and its two reflex neighbors. □

Lemma 5 *In an IUP, we can find a set of at most 8 edges that contains the 4 tab edges in $O(n^4)$ time.*

Proof. Compute the 4 maximal cliques from Lemma 4. These cliques have exactly 3 convex vertices each, and tab edges are incident to two convex vertices, narrowing our choice of tab down to $4 \cdot \binom{3}{2} = 12$ edges. We detect and remove any vertical or horizontal non-boundary edges by checking which edges are in multiple maximal cliques. There are 4 of these; thus, we have 8 edges to consider. □

Furthermore these eight edges form 4 disjoint paths on 2 edges, and the middle vertex on each path is a tab vertex. Lastly, we note that these known tab vertices are on the long staircases. We now show how to eliminate the remaining 4 non-tab edges.

Lemma 6 *We can compute the 4 tab edges of an IUP in $O(n^4)$ time.*

Proof. First find the 8 candidates as in Lemma 5.

Recall that we already know one vertex on each tab edge, and that these vertices are on the long staircases. Let one of them be called u. Now it remains the find u's neighbor on its tab edge. Vertex u has two candidate neighbors; let's call them v and w. Just for concreteness, let's say u is on the north edge and is on the northeast staircase.

Suppose w.l.o.g. that v has more reflex neighbors than w, then v is u's neighbor on a tab edge, because it sees reflex vertices on the whole northeast and southeast staircases, while w sees only a subset of those. Otherwise v and w have the same number of reflex neighbors. Then either v or w has more convex neighbors. Suppose w.l.o.g. that v is a tab vertex, then v has fewer convex neighbors than w. To see why, note that since u is on northeast (long) staircase, v is on the northwest (short) staircase. Vertex v has convex neighbors u, w, and every convex vertex on the southeast (short) staircase. Likewise, w has convex neighbors v, u, every convex vertex on the northeast (long) staircase (including u) and one vertex on the southwest (long) staircase.

We can do these checks for all such pairs v and w, giving us all tab edges. □

Now that we have the 4 tab edges, we pick one arbitrarily to be the north edge. We show how to orient the polygon such that the northwest staircase is short and the northeast staircase is long. We do this by identifying the convex vertices on the short staircase by computing *elementary cliques*.

Definition 2 (elementary clique) *An* elementary clique *in an IUP is a maximal clique that contains exactly 3 convex vertices and either contains a tab edge or no tab vertices. (See Figure 2.)*

Lemma 7 *We can identify the elementary cliques with vertices on the northwest staircase in $O(n^4)$ time.*

Proof. First, we compute the $O(n)$ elementary cliques as follows. We compute all axis-aligned, non-boundary visibility edges that have only convex endvertices. We then compute the 2 maximal cliques containing each such edge, and keep only the cliques with 3 convex vertices.

Let C_0 be the unique maximal (elementary) clique containing the north edge. Then C_0 contains 4 reflex vertices R_0, 3 of which are in exactly one other elementary clique, which we'll call C_1. The northwest staircase has k_{NW} elementary cliques, $C_0, \ldots, C_{k_{NW}-1}$, where each clique C_i contains reflex vertices R_i. Then $|R_i \cap C_{i+1}| = 3$, and for $j \neq i, i+1$ $R_i \cap C_j = \emptyset$.

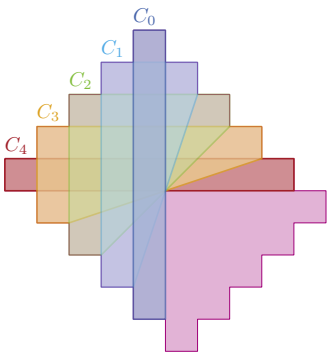


Figure 2: Elementary cliques of an IUP. Note that only half of the elementary cliques are shown.

Therefore, from elementary clique C_i, we can compute elementary clique C_{i+1} by searching for the only other elementary clique containing reflex vertices R_i. Once we reach an elementary clique containing a tab edge, then we have computed all elementary cliques on the northwest staircase. This tab edge is the west edge and we are finished. □

We now show how to assign the convex vertices from the elementary cliques to each staircase.

Lemma 8 *We can identify the convex vertices on the northwest staircase in $O(n^4)$ time.*

Proof. First we assign all non-tab convex vertices. Let $\mathcal{C}_{\text{NW}}$ be a non-primary elementary clique containing (non-tab) convex vertices v_{NW}, v_{NE} and v_{SW} from the northwest, northeast and southwest staircases respectively. Then v_{NW} sees neither a vertex on the north nor west edge. However, v_{NE} sees a vertex on the west edge, and v_{SW} sees a vertex on the north edge. Therefore, we can compute the non-tab convex vertices on the northwest staircase by checking which vertices have no neighbors on the north or west edges. Now we assign the tab vertices to a staircase. There is exactly one visibility edge connecting a vertex v_N on the north edge to a vertex v_W on the west edge. Then v_W is on the southeast staircase, v_N is on the northeast staircase, and the remaining two tab vertices are on the northwest staircase.

Furthermore, we can assign the remaining convex vertices to the southwest and northeast staircases: Convex vertices on the southwest (northeast) staircase cannot see vertices on the west (north) edge. □

We can repeat the above algorithm to find the convex vertices on the southeast staircase. However, we

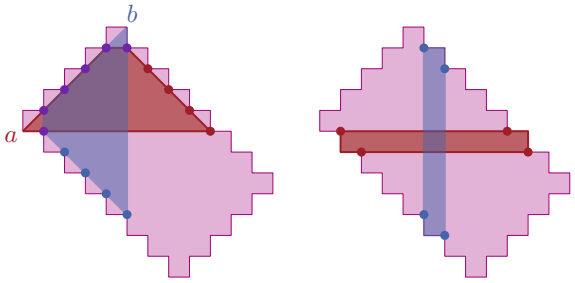


Figure 3: Left: Tab vertices a and b see unique reflex vertices on long staircases. Right: We assign the remaining reflex vertices with rectangles between long staircases.

still need to assign any remaining vertices to the middle of the southwest and northeast (long) staircases. Note that these are the vertices which cannot see the northwest and southeast staircases and, therefore, weren't assigned in the above algorithm.

Lemma 9 *We can assign the convex vertices to their long staircases in $O(n^4)$ time.*

Proof. Let $\mathcal{W}$ and $\mathcal{E}$ be all the convex vertices on the southwest and northeast staircases (which we are computing) and let $\mathcal{W}_0$ and $\mathcal{E}_0$ be the convex vertices on the southwest and northeast staircases that are already known from the elementary cliques from Lemma 8. Let $N_c(v)$ denote the set of convex neighbors on the opposite staircase of some vertex v. Then, for each vertex $w_0 \in \mathcal{W}_0$, $N_c(w_0) \subseteq \mathcal{E}$, i.e., the convex neighbors of the (convex) vertices in $\mathcal{W}_0$ are on the northeast staircase. Similarly, for each vertex $e_0 \in \mathcal{E}_0$, $N_c(e_0) \subseteq \mathcal{W}$. Then we can iteratively define sets $\mathcal{E}_i = \mathcal{E}_{i-1} \setminus \cup_{w \in \mathcal{W}_{i-1}} N_c(w)$ and $\mathcal{W}_i = \mathcal{W}_{i-1} \setminus \cup_{e \in \mathcal{E}_{i-1}} N_c(e)$ and identify all vertices of the southwest and northeast staircases as $\mathcal{W} = \cup_i \mathcal{W}_i$ and $\mathcal{E} = \cup_i \mathcal{E}_i$.

To order the vertices along the southwest staircase, note that the sets $\mathcal{W}_i$ should appear in order of increasing i from top to bottom. Also note that if one were to assign the vertices of W_i the staircase from top to bottom, each vertex w_i in this order would see fewer vertices of $\mathcal{E}_{i-1}$. Thus, we can order the vertices within each $\mathcal{W}_i$. The argument for ordering vertices of $\mathcal{E}_i$ is symmetric. □

We can now choose the south and east edges: a vertex on the east (south) edge can see convex vertices on the southwest (northeast) staircase.

Lemma 10 *We can assign the reflex vertices to each staircase in $O(n^4)$ time.*

Proof. Once the convex vertices are ordered on the staircases, we can compare the reflex vertices that are seen from the tab vertices. Let a and b be vertices on different tab edges, that are visible along a short staircase (see Figure 3, left), and let $\mathcal{R}$ be the set of all reflex vertices of the IUP, and $N(v)$ be a set of all neighbors of vertex v in the visibility graph of IUP. Then $\mathcal{R}_0 = N(a) \cap N(b) \cap \mathcal{R}$ contains all reflex vertices from the short staircase, plus two extra reflex vertices from the neighboring long staircases. The remaining vertices $N(a) \setminus \mathcal{R}_0$ are on one long staircase (and $N(b) \setminus R_0$ are on the other long staircase).

Thus, we can find many reflex vertices on the long staircases, except the end vertices and potentially those in the middle of the staircases. To find the remaining ones, we build rectangles (maximal cliques) consisting of two convex vertices u and v on the opposite staircases and a known reflex vertex w, such that (u, w) forms a boundary edge of the IUP (see Figure 3, right). These rectangles define new reflex vertices on the opposite staircase from w. Thus, we iteratively discover all new reflex vertices. □

Lemma 11 *We can order the reflex vertices on each staircase in $O(n^4)$ time.*

Proof. Let $c_0, \ldots, c_k$ be the convex vertices in order on some staircase S containing reflex vertices R. Then $N(c_i) \cap N(c_{i+1}) \cap R = \{r_i\}$ where r_i is the reflex vertex between c_i and c_{i+1} on staircase S. Thus, we know the order of the reflex vertices along each staircase. □

Therefore, we have ordered the vertices on all staircases, constructing the Hamiltonian cycle of boundary edges in G_P, arriving at the following theorem:

Theorem 12 *In $O(n^4)$ time, we can reconstruct an IUP P from its visibility graph G_P.*

References

[1] J. Abello, Ö. Eğecioğlu, and K. Kumar. Visibility graphs of staircase polygons and the weak Bruhat order, I: From visibility graphs to maximal chains. *Discrete Comput. Geom.*, 14(3):331–358, 1995.

[2] S.-H. Choi, S. Y. Shin, and K.-Y. Chwa. Characterizing and recognizing the visibility graph of a funnel-shaped polygon. *Algorithmica*, 14(1):27–51, 1995.

[3] H. Everett. *Visibility graph recognition.* PhD thesis, University of Toronto, 1990.

[4] H. Everett and D. Corneil. Recognizing visibility graphs of spiral polygons. *J. Algorithms*, 11(1):1–26, 1990.

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

[6] L. Jackson and S. Wismath. Orthogonal polygon reconstruction from stabbing information. *Comp. Geom.-Theor. Appl.*, 23(1):69–83, 2002.

[7] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, 1987.

An Approximation Algorithm for the Art Gallery Problem

Édouard Bonnet * Tillmann Miltzow †

Abstract

Given a simple polygon $\mathcal{P}$ on n vertices, two points x, y in $\mathcal{P}$ are said to be *visible* to each other if the line segment between x and y is contained in $\mathcal{P}$. The *point-guard art gallery problem* asks for a minimum set S such that every point in $\mathcal{P}$ is visible from a point in S. Assuming integer coordinates and a special general position assumption, we present the first $O(\log \mathrm{OPT})$-approximation algorithm for the point guard art gallery problem. This algorithm combines ideas of Efrat and Har-Peled [7] and Deshpande et al. [3, 4]. In addition, we point out a mistake in the latter.

1 Introduction

In 1973, Victor Klee posed to Chvátal the art gallery problem as follows. Given a simple polygon $\mathcal{P}$ on n vertices, two points x, y in $\mathcal{P}$ are said to be visible to each other if the line segment between x and y is contained in $\mathcal{P}$. The point-guard art gallery problem asks for a minimum set S such that every point in P is visible from a point in S.

A huge amount of research is committed to the studies of combinatorial and algorithmic aspects of the art gallery problem, see the following surveys [10, 18–20]. Most of this research, however is not focused directly on the art gallery problem but on variants, based on different definitions of visibility, restricted classes of polygons, different shapes and positions of guards and so on. The arguably most natural definition of visibility is the one we defined above. One of the first combinatorial results is the elegant proof of Fisk that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary for a polygon with n vertices [9].

On the algorithmic side, very few variants are solvable in polynomial time [5, 17], but most results are on approximating the minimum number of guards [3, 4, 7, 11, 14, 15]. Many of the approximation algorithms are based on the fact that the range space defined by the visibility regions has bounded VC-dimension [12, 13, 21] for simple polygons. This makes it easy to use the algorithmic ideas of Clarkson [1, 2].

On the lower bound side the paper of Eidenbenz et al. showed for most relevant variants NP-hardness and inapproximability [8]. In particular, they show for the main variants that there is no c-approximation algorithm for simple polygons, for some constant c. For polygons with holes, they can even show that there is no $o(\log n)$-approximation algorithm.

Very surprisingly, there is only one exact algorithm for the point guard art gallery problem running in $n^{O(k)}$ time attributed to Micha Sharir [7]. (Here, k is the size of the optimal solution.) And only, we recently the authors could give an almost matching lower bound by ruling out $n^{o(k/\log k)}$, assuming ETH [6].

Regarding approximation algorithms for the point guard variant, the results are very sparse. For monotone polygons and rectilinear polygons approximation algorithms are known [16]. For general polygons, Deshpande et al. gave a randomized pseudopolynomial time $O(\log n)$ approximation algorithm [3, 4]. However, we show that their algorithm is not correct. Efrat and Har-Peled gave a randomized polynomial time algorithm, by restricting guards to a very fine grid. They show that their grid solution S_{grid} is at most by a factor of $\log(OPT_{\text{grid}})$ away from the optimal grid solution OPT_{grid}. However, they could not prove that their OPT_{grid} is indeed an approximation of an optimal guard placement OPT. Developing the ideas of Deshpande et al. in combination of the algorithm of Efrat and Har-Peled we attain the first randomized polynomial time approximation algorithm for general simple polygons.

To keep the proof simple, we introduce a *special general position assumption*. We say a line is an *extension* of a polygon if it is the supporting line of two vertices of the polygon $v, w \in V(\mathcal{P})$. We say a polygon satisfies the special general position assumption, if no three points lie on a line and no three extensions meet in a point $p \in \mathcal{P} \setminus V(\mathcal{P})$.

Theorem 1 *There is an $O(\log |OPT|)$ approximation algorithm for* Point Guard Art Gallery *that runs in randomized polynomial time in the size of the input, given the following assumption:*

integer vertex representation: *Vertices are given by integers, represented in binary.*

*Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), edouard.bonnet@lamsade.dauphine.fr

†Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), t.miltzow@gmail.com

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

special general position assumption: *No three extensions meet in a point $p \in \mathcal{P} \setminus V(\mathcal{P})$.*

In this four page abstract we focus on the key ideas and henceforth omit the proofs of most lemmas and some technical details.

2 Preliminaries

Polygons and visibility. For any two distinct points v and w in the plane, we denote by $\text{seg}(v, w)$ the segment whose two endpoints are v and w, by $\text{ray}(v, w)$ the ray starting at v and passing through w, by $\ell(v, w)$ the supporting line passing through v and w. We also denote by $\text{disk}(v, r)$ the disk centered in point v and whose radius is r, and by $\text{dist}(a, b)$ the distance between object a and object b.

A polygon is *simple* if it is not self-crossing and has no holes. For any point x in a polygon $\mathcal{P}$, $V(x)$ denotes the *visibility region* of x within $\mathcal{P}$, that is the set of all the points $y \in \mathcal{P}$ such that segment $\text{seg}(x, y)$ is entirely contained in $\mathcal{P}$.

3 Approximation

Given a polygon $\mathcal{P}$, we will always assume that all its vertices are given by *positive* integers in binary. We denote by M the largest integer, by D the diameter of the polygon and define $L = 10D$. Note that this implies $L \geq 5M$ and $\log L$ is linear in the input size. We set the width to $w = L^{-20}$ and define the grid Γ as $(w \cdot \mathbb{Z}^2) \cap \mathcal{P}$. Note that all vertices of $\mathcal{P}$ have integer coordinates and thus are included in Γ. We denote by OPT an optimal solution to the point guard art gallery problem and by k its size. We denote by OPT_{grid} an optimal solution to the grid guard art gallery problem, this is, guards are restricted to lie on the grid Γ, and denote by $k_{\text{grid}} = |OPT_{grid}|$.

The idea is to show that the algorithm of Efrat and Har-Peled gives an approximation algorithm under the integer vertex representation assumption, the grid Γ as described above and the special general position assumption.

Theorem 2 (Efrat, Har-Peled [7]) *Given a simple polygon $\mathcal{P}$ with n vertices, one can spread a grid Γ inside $\mathcal{P}$, and compute an $O(\log k_{\text{grid}})$-approximation to the smallest subset of Γ that sees $\mathcal{P}$. The expected running time of the algorithm is $O(n\, k_{\text{grid}}^2 \log k_{grid} \log(nk_{\text{grid}}) \log^2 \Delta)$, where Δ is the ratio between the diameter of the polygon and the grid size.*

Note that the grid size equals $w = L^{-20}$, thus $\Delta \leq L^{21}$ and consequently $\log \Delta \leq 21 \log L$, which is linear in the size of the input. It remains to show the following lemma given the assumptions and notation mentioned above.

Lemma 3 *$\exists c \in \mathbb{N}$ such that $k_{\text{grid}} \leq c \cdot k$.*

The way we use the integer coordinate assumption is to infer distance lower bounds between various objects of interest.

Lemma 4 (Distances) *Let v and w be vertices of $\mathcal{P}$, ℓ_1 and ℓ_2 supporting lines of two vertices, and p and q intersections of supporting lines. Then the following holds:*

1. *$dist(v, w) > 0 \Rightarrow dist(v, w) \geq 1$.*
2. *$dist(v, \ell_1) > 0 \Rightarrow dist(v, \ell_1) \geq 1/L$.*
3. *$dist(p, \ell_1) > 0 \Rightarrow dist(p, \ell_1) \geq 1/L^5$.*
4. *$dist(p, q) > 0 \Rightarrow dist(p, q) \geq 1/L^4$.*
5. *Let $\ell_1 \neq \ell_2$ be parallel. Then $dist(\ell_1, \ell_2) \geq 1/L$.*
6. *Let $a \in \mathcal{P}$ be a point and ℓ_1 and ℓ_2 be some lines with $dist(\ell_i, a) < d$, for $i = 1, 2$. Then ℓ_1 and ℓ_2 intersect in a point p with $dist(a, p) \leq dL^2$.*

Proof. The idea of the proof is very simple. We look up the formula for each claimed distance. This formula is in most cases a fraction. By the assumption the nominator is at least one. The variables in the denominator can be safely upper bounded by L. The claim follows immediately. □

Grid points. See Figure 1 for the following description. Each point x of the optimal solution is in some grid cell $\text{grid}(x)$. For the sake of brevity, we assume that the grid cell does not contain any point of the boundary of the polygon, as in Figure 1 b) and c).

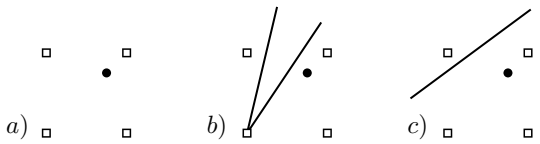


Figure 1: The way that the polygon boundary might interact with the grid cell.

Local visibility containment property. We say a point x in the grid cell formed by g_1, g_2, g_3, g_4 has the *local visibility containment property* (LVCP) if $V(x) \subseteq V(g_1) \cup V(g_2) \cup V(g_3) \cup V(g_4)$.

Cones. Given a point x and two points r_1, r_2 in the plane, we define the cone of x with respect to r_1 and r_2 as the unique cone $C(x)$ with apex x that is bounded by $\text{ray}(x, r_1)$ and $\text{ray}(x, r_2)$ and forms an angle smaller than π.

Opposite reflex vertices and bad regions. Given a polygon $\mathcal{P}$ and two reflex vertices r_1 and r_2, consider the supporting line $\ell = \ell(r_1, r_2)$ restricted to $\mathcal{P}$. The supporting line defines two halfplanes ℓ^+ and ℓ^-. We say r_1 is *opposite* to r_2 if $\text{disk}(r_1, \varepsilon) \cap \partial\mathcal{P} \subset \ell^+$ and $\text{disk}(r_2, \varepsilon) \cap \partial P \subset \ell^-$ for some $\varepsilon > 0$.

Given two opposite reflex vertices r_1 and r_2, we define their *bad regions* as the stripe around $\ell(r_1, r_2)$ with width $8wL$, see Figure 2.

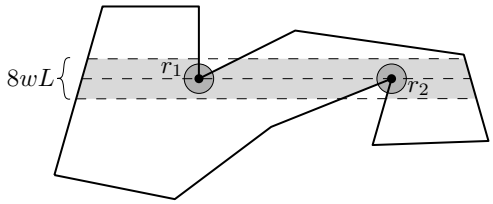


Figure 2: Illustration of opposite reflex vertices and bad regions.

Lemma 5 (Loc. Visib. Containment Prop.) *Let $x \in \mathcal{P}$ be outside any bad region then x has the local visibility property.*

Proof. Here, we give only the idea. Let $x \in \mathcal{P}$ be some point and g_1, g_2, g_3, g_4 be the grid points surrounding x. Further let r_1 and r_2 be any two reflex vertices visible from x. the same fashion. We can restrict ourselves to show $C(x) \subseteq C(g_1) \cup C(g_2) \cup C(g_3) \cup C(g_4)$. Further, we only have to show this for the region behind $\ell(r_1, r_2)$. In case that there is some $g_i \in C(x)$, this is trivially true. For the other case, see Figure 3. In Figure 3, we see that the dark red

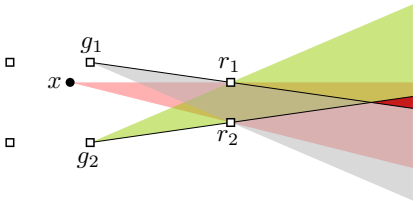


Figure 3: The cones $C(g_1)$ and $C(g_2)$ are not covering the same area as $C(x)$, behind $\ell(r_1, r_2)$. The reason is that $\text{ray}(g_1, r_2)$ and $\text{ray}(g_2, r_2)$ intersect.

area of $C(x)$ is not covered by by $C(g_1)$ and $C(g_2)$. It is not so difficult to see that it is sufficient to show that this situation will not appear. For this purpose it is sufficient to show that $\text{ray}(g_1, r_1)$ and $\text{ray}(g_2, r_2)$ do not cross. Roughly speaking, we compare the distance between the rays close to x and close to the reflex vertices. If the distance between the rays increases, we know that they will never meet. Very helpful for us is the distance of at least one between the reflex vertices and that x is outside of any bad region. □

It is easy to believe that the local visibility containment property also holds inside the bad region. However this is not true. Here, we will briefly describe a counter-example. In particular, this example shows that the algorithm of Deshpande et al. is not correct as it is stated. However, we want to mention that their paper has ideas that motivated the algorithm presented in this preprint.

Deshpande et al. described an algorithm that worked in several steps [3, 4]. In the first step they generate a large number of points P that they store explicitly in memory. Thereafter, they find a solution for the point guard art gallery problem $S \subset P$. The crucial point is the claim that their point set P satisfies some variant of local visibility containment property. To be more precise, we say a set of points P has the *general local visibility containment property*, if for every point $x \in \mathcal{P}$ exists a finite collection $C \subset P \cap \text{disk}(x, 1)$, such that $V(x) \subseteq \bigcup_{p \in C} V(p)$. Our example shows that it is impossible to attain any finite point set that has this property.

Example See Figure 4, for the following description.

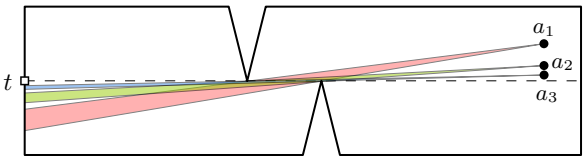


Figure 4: Illustration of the counter-example to the algorithm of Deshpande et al. [3, 4].

We have two opposite reflex vertices with supporting line ℓ. The points $(a_i)_{i \in \mathbb{N}}$ are chosen closer and closer to ℓ on the right side of the polygon. None of the a_i can see t, as this would require to be actually on ℓ. We choose the points $(a_i)_{i \in \mathbb{N}}$ in a way that their intervals will be all disjoint and arbitrarily close to t.

Consider now any finite set of points C in the "vicinity" of the $(a_i)_{i \in \mathbb{N}}$. We will show that there is some a_i, which sees some interval close to t, that is not seen by any point in C. Recall that no point sees the entire interval around t, but the visibility of the a_is come arbitrarily close to t. Thus, there is some a_i that sees something that is not visible by any point in C.

This shows that the general local visibility containment property cannot be attained already in this fairly straightforward polygon.

Despite the fact that it is not possible to achieve a general local visibility containment property for all points in $\mathcal{P}$, the exceptions are only for points in the bad regions. These cases we can handle in a different manner.

Lemma 6 *Let $x \in \mathcal{P}$ in three or more bad regions.*

Then there exists a reflex vertex r with $dist(x, r) \leq wL^3$. And r is one of the defining reflex vertices for all bad regions that x belongs to.

Proof. Let ℓ_1, ℓ_2, ℓ_3 be three different extensions. And we assume that x is in the bad region of all three of them. In case they all have on reflex vertex in common, it must be a defining reflex vertex and x cannot be too far away from r.

So assume that they do not have a reflex vertex in common. We want to show that the intersection of their corresponding bad regions is empty. These three lines form a triangle Δ with vertices of pairwise distance at least L^{-4} by Lemma 4 Item 4. Assume for the purpose of contradiction that x is in the bad region of all three lines. Then x has distance at most $4wL$ to all lines. By Lemma 4 Item 6 x has distance at most $4wL^3$ to the vertices of Δ. By the triangle inequality the vertices have pairwise distance at most $8wL^3 \ll L^{-4}$ – a contradiction. □

Proof. [Lemma 3] Let OPT be an optimal solution of size k. We construct a set of $6k$ guards $G \subset \Gamma$, which see the entire polygon. For each guard $x \in OPT$, we add the four grid points of x into G. Further, if x is in one or two bad regions, add the corresponding reflex vertex for each bad region into G. In case x is in more than two bad regions there is one reflex vertex that is defining all of them. Add it to G. For each $x \in OPT$ the local containment property holds, except for the bad regions it is in. These parts are seen by the reflex vertices we added. □

Acknowledgments

Both authors are supported by the ERC grant PARAMTIGHT: "Parameterized complexity and the search for tight complexity results", no. 280152. We thank Meirav Zehavi and Saeed Merhabi for useful discussions during a preliminary stage of the project. We also want to thank anonymous reviewers for their helpful feedback.

References

[1] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[2] K. L. Clarkson. Algorithms for polytope covering and approximation. In *WADS 1993*, pages 246–252, 1993.

[3] A. Deshpande. A pseudo-polynomial time $O(\log^2 n)$-approximation algorithm for art gallery problems. Master's thesis, Department of Mechanical Engineering, Department of Electrical Engineering and Computer Science, MIT, 2006.

[4] A. Deshpande, T. Kim, E. D. Demaine, and S. E. Sarma. A pseudopolynomial time $O(\log n)$-approximation algorithm for art gallery problems. In *WADS 2007*, pages 163–174, 2007.

[5] S. Durocher and S. Mehrabi. Guarding orthogonal art galleries using sliding cameras: algorithmic and hardness results. In *MFCS 2013*, pages 314–324. Springer, 2013.

[6] Édouard Bonnet and T. Miltzow. Parameterized hardness of art gallery problems. In *EuroCG 2016*, page to appear, 2016.

[7] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

[8] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[9] S. Fisk. A short proof of chvátal's watchman theorem. *J. Comb. Theory, Ser. B*, 24(3):374, 1978.

[10] S. K. Ghosh. *Visibility algorithms in the plane*. Cambridge University Press, 2007.

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

[12] A. Gilbers and R. Klein. A new upper bound for the vc-dimension of visibility regions. *Computational Geometry*, 47(1):61–74, 2014.

[13] G. Kalai and J. Matoušek. Guarding galleries where every point sees a large area. *Israel Journal of Mathematics*, 101(1):125–139, 1997.

[14] J. King. Fast vertex guarding for polygons with and without holes. *Comput. Geom.*, 46(3):219–231, 2013.

[15] D. G. Kirkpatrick. An $O(lglgOPT)$-approximation algorithm for multi-guarding galleries. *Discrete & Computational Geometry*, 53(2):327–343, 2015.

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

[17] R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *J. Comput. Syst. Sci.*, 40(1):19–48, 1990.

[18] J. O'rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[19] T. C. Shermer. Recent results in art galleries [geometry]. *Proceedings of the IEEE*, 80(9):1384–1399, 1992.

[20] J. Urrutia et al. Art gallery and illumination problems. *Handbook of computational geometry*, 1(1):973–1027, 2000.

[21] P. Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104(1):1–16, 1998.

Parameterized Hardness of Art Gallery Problems

Édouard Bonnet * Tillmann Miltzow †

Abstract

Given a simple polygon $\mathcal{P}$ on n vertices, two points x, y in $\mathcal{P}$ are said to be visible to each other if the line segment between x and y is contained in $\mathcal{P}$. The point guard art gallery problem asks for a minimum set S such that every point in $\mathcal{P}$ is visible from a point in S. The vertex guard art gallery problem asks for such a set S subset of the vertices of $\mathcal{P}$. The set S is referred to as guards. We show $W[1]$-hardness of both variants, when parameterized by the number k of guards. We even rule out any $n^{o(k/\log k)}$ algorithm under the exponential time hypothesis.

1 Introduction

Given a simple polygon $\mathcal{P}$ on n vertices, two points x, y in $\mathcal{P}$ are said to be visible to each other if the line segment between x and y is contained in $\mathcal{P}$. The point-guard art gallery problem asks for a minimum set S of points called *guards* such that every point in P is visible from a point in S. The vertex guard art gallery problem asks for such a set of guards S subset of the vertices of $\mathcal{P}$.

One of the first combinatorial results is the elegant proof of Fisk that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary for a polygon with n vertices [8]. On the algorithmic side, very few variants are solvable in polynomial time [5, 11], but most results are on approximating the minimum number of guards [3, 4, 6, 9]. On the lower bound side the paper of Eidenbenz et al. showed for most relevant variants NP-hardness and inapproximability [7]. In particular, their reduction from Set-Cover implies that the art gallery is $W[2]$-hard on polygons with holes and that there is no $n^{o(k)}$ algorithm, to determine if k guards are sufficient for a given gallery with n vertices, under the exponential time hypothesis [7, Sec.4]. However, polygons with holes are very different to simple polygons as they have unbounded VC-dimension [12]. In particular none of these reductions rule out a fixed parameter tractable algorithm (i.e., whose running time is $O(f(k)n^c)$ where f is any computable function and c is a constant) for simple polygons (see [2] for an introduction to parameterized complexity.).

Obviously, the vertex guard variant can be solved in time $O(n^{k+2})$ by trying out all possible subsets of size k of the vertices and checking if one of those subsets sees the whole polygon. Not obvious at all is the algorithm running in time $n^{O(k)}$ for the point guard variant using standard tools from real algebraic geometry [1]. Despite the fact that the first algorithm is extremely basic and the second algorithm, even with remarkably sophisticated tools, uses almost no problem specific insights, no better exact parameterized algorithms are known.

We present the first conditional lower bounds for the parameterized art gallery problem for simple polygons:

Theorem 1 (Point guard hardness) Point Guard Art Gallery *parameterized by the number of guards k is $W[1]$-hard, and is not solvable in time $n^{o(k/\log k)}$, under the ETH.*

Theorem 2 (Vertex guard hardness) Vertex Guard Art Gallery *is $W[1]$-hard, and is not solvable in time $n^{o(k/\log k)}$, under the ETH.*

2 Preliminaries

For any two integers $x < y$, we set $[x, y] := \{x, x+1, \ldots, y-1, y\}$, and for any positive integer x, $[x] := [1, x]$. The *Exponential Time Hypothesis* (ETH) is a conjecture by Impagliazzo et al. [10] asserting that there is no $2^{o(n)}$-time algorithm for 3-SAT on instances with n variables.

Polygons and visibility. For any two distinct points v and w in the plane, we denote by $\text{seg}(v, w)$ the segment whose two endpoints are v and w, by $\text{ray}(v, w)$ the ray starting at v and passing through w, by $\ell(v, w)$ the supporting line passing through v and w.

A polygon is *simple* if it is not self-crossing and has no holes. For any point x in a polygon $\mathcal{P}$, $V_{\mathcal{P}}(x)$, or simply $V(x)$, denotes the *visibility region* of x within $\mathcal{P}$, that is the set of all the points $y \in \mathcal{P}$ such that segment $\text{seg}(x, y)$ is entirely contained in $\mathcal{P}$. We say that two vertices v and w of a polygon $\mathcal{P}$ are *neighbors* or *consecutive* if vw is an edge of $\mathcal{P}$. A *subpolygon* $\mathcal{P}'$ of a simple polygon $\mathcal{P}$ is defined by any l distinct consecutive vertices $v_1, v_2, \ldots, v_l$ of $\mathcal{P}$ (that is, for every

*Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), edouard.bonnet@lamsade.dauphine.fr

†Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), t.miltzow@gmail.com

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

$i \in [l-1]$, v_i and v_{i+1} are neighbors in $\mathcal{P}$) such that v_1v_l does not cross any edge of $\mathcal{P}$.

Given a vertex v and two points p and p', we call *triangular pocket rooted at vertex v and supported by ray(v,p) and ray(v,p')* a sub-polygon w, v, w' such that ray(v,w) passes through p, ray(v,w') passes through p'. We say that v is the *root* of the triangular pocket that we denote $\mathcal{P}(v)$. We also say that the pocket $\mathcal{P}(v)$ *points* towards p and p'.

Structured 2-Track Hitting Set. We introduce a new problem which will constitute a handy starting point to show Theorem 1 and 2. In the 2-Track Hitting Set problem, the input consists of an integer k, two sets A and B of the same cardinality totally ordered by $\leq_A$ and $\leq_B$, and two sets $\mathcal{S}_A$ of A-intervals (that is a set of consecutive elements of A according to $\leq_A$), and $\mathcal{S}_B$ of B-intervals. In addition, the elements of A and B are in one-to-one correspondence $\phi : A \to B$ and each pair $(a, \phi(a))$ is called a 2-*element*. The goal is to find a set S of k 2-elements such that the first projection of S is a hitting set of A, and the second projection of S is a hitting set of B. Structured 2-Track Hitting Set is the same problem with color classes over the 2-elements, and a restriction on the one-to-one mapping ϕ. A is partitioned into k classes $(C_1, C_2, \ldots, C_k)$ where $C_j = \{a_1^j, a_2^j, \ldots, a_t^j\}$ for each $j \in [k]$, where $|A| = tk$, and is ordered: $a_1^1, a_2^1, \ldots, a_t^1, a_1^2, a_2^2, \ldots, a_t^2, \ldots, a_1^k, a_2^k, \ldots, a_t^k$. We define $C'_j := \phi(C_j)$ and $b_i^j := \phi(a_i^j)$ for all $i \in [t]$ and $j \in [k]$. We now impose that ϕ is such that, for each $j \in [k]$, the t elements of C'_j are consecutive along $\leq_B$. That is, B is ordered: $C'_{\sigma(1)}, C'_{\sigma(2)}, \ldots, C'_{\sigma(k)}$ for some permutation on $[k]$, $\sigma \in \mathfrak{S}_k$. For each $j \in [k]$, the order of the elements within C'_j can be described by a permutation $\sigma_j \in \mathfrak{S}_t$ such that the ordering of C'_j is: $b^j_{\sigma_j(1)}, b^j_{\sigma_j(2)}, \ldots, b^j_{\sigma_j(t)}$. Due to space limitations, we omit the proof of the following theorem.

Theorem 3 Structured 2-Track Hitting Set *is $W[1]$-hard, and not solvable in time $|\mathcal{I}|^{o(k/\log k)}$, unless the ETH fails.*

3 Point Guard

Overview of the reduction. Given an instance $\mathcal{I} = (k \in \mathbb{N}, t \in \mathbb{N}, \sigma \in \mathfrak{S}_k, \sigma_1 \in \mathfrak{S}_t, \ldots, \sigma_k \in \mathfrak{S}_t, \mathcal{S}_A, \mathcal{S}_B)$, we build a simple polygon $\mathcal{P}$ with $O(kt + |\mathcal{S}_A| + |\mathcal{S}_B|)$ vertices, such that $\mathcal{I}$ is a YES-instance iff $\mathcal{P}$ can be guarded by $3k$ points.

The global strategy of the reduction is to *allocate*, for each color class $j \in [k]$, $2t$ special points in the polygon $\alpha_1^j, \ldots, \alpha_t^j$ and $\beta_1^j, \ldots, \beta_t^j$. Placing a guard in α_i^j (resp. β_i^j) shall correspond to picking a 2-element whose first (resp. second) component is a_i^j (resp. b_i^j).

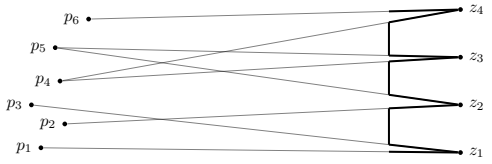


Figure 1: Interval gadgets encoding $\{p_1, p_2, p_3\}$, $\{p_2, p_3, p_4, p_5\}$, $\{p_4, p_5\}$, and $\{p_4, p_5, p_6\}$.

The points α_i^j's and β_i^j's ordered by increasing y-coordinates will match the order of the a_i^j's along $\leq_A$ and then of the b_i^j's along $\leq_B$. Then, far in the horizontal direction, we will place pockets to encode each A-interval of $\mathcal{S}_A$, and each B-interval of $\mathcal{S}_B$ (see Figure 1).

The critical issue will be to *link* point α_i^j to point β_i^j. Indeed, in the Structured 2-Track Hitting Set problem, one selects 2-elements (one per color class), so we should prevent one from placing two guards in α_i^j and $\beta_{i'}^j$ with $i \neq i'$. Due to a technicality, we will introduce a *copy* $\overline{\alpha}_i^j$ of each α_i^j. In each part of the gallery encoding a color class $j \in [k]$, the only way of guarding all the pockets with only three guards is to place them in α_i^j, $\overline{\alpha}_i^j$, and β_i^j for some $i \in [t]$. Hence, $3k$ guards will be necessary and sufficient to guard the whole $\mathcal{P}$ iff there is a solution to the instance of Structured 2-Track Hitting Set.

We now sketch the construction.

Allocated points and interval gadgets. The position of the α_i^j's and β_i^j can be seen on Figure 2 and Figure 4. It is such that the ordering of the α_i^j's (resp. β_i^j) by increasing y-coordinate matches the order $\leq_A$ on the a_i^j's (resp. $\leq_B$ on the b_i^j's). Also, α_i^j and β_i^j shares the same x-coordinate for each $j \in [k], i \in [t]$. There is a quite large gap D along the x-axis between a point α_t^j and α_t^{j+1}.

For each A-interval $I_q = [a_i^j, a_{i'}^{j'}] \in \mathcal{S}_A$, we put, at a very large distance F to the right of the α_i^j's, one triangular pocket $\mathcal{P}(z_{A,q})$ rooted at vertex $z_{A,q}$ and supported by ray$(z_{A,q}, \alpha_i^j)$ and ray$(z_{A,q}, \alpha_{i'}^{j'})$. This way, the only $\alpha_{i''}^{j''}$ seeing vertex $z_{A,q}$ are all the points such that $a_i^j \leq_A a_{i''}^{j''} \leq_A a_{i'}^{j'}$ (see Figure 1 and Figure 4). We do the same for the B-intervals.

Weak linkers. We now describe how we *link* each point α_i^j to its associate β_i^j. See Figure 2 for a description of the following *weak linker* gadget.

For each $j \in [k]$, let us mentally draw ray(α_t^j, β_1^j) and consider points slightly to the left of this ray and quite far. Let us call $\mathcal{R}^j_{\text{left}}$ that informal region of points. Any point in $\mathcal{R}^j_{\text{left}}$ sees, from right to left, in this order α_1^j, α_2^j up to α_t^j, and then,

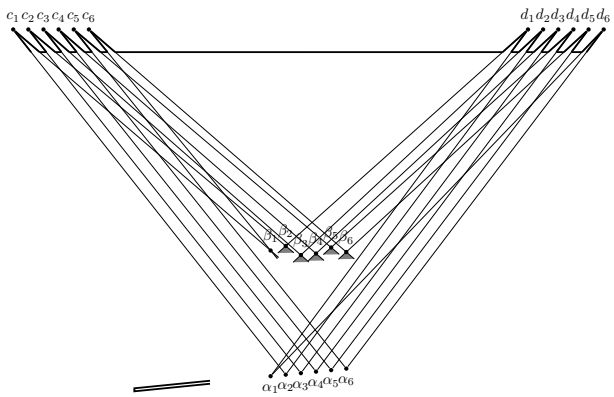


Figure 2: Weak point linker gadget.

β_1^j, β_2^j up to β_t^j. In $\mathcal{R}_{\text{left}}^j$, for each $i \in [t-1]$, we place a triangular pocket $\mathcal{P}(c_i^j)$ rooted at vertex c_i^j and supported by $\text{ray}(c_i^j, \alpha_{i+1}^j)$ and $\text{ray}(c_i^j, \beta_i^j)$. We place also a triangular pocket $\mathcal{P}(c_t^j)$ rooted at c_t^j supported by $\text{ray}(c_i^j, \beta_1^j)$ and $\text{ray}(c_i^j, \beta_t^j)$. We place mirroring pockets $P(d_1^j), \ldots, \mathcal{P}(d_t^j)$ in the region slightly to the right of $\text{ray}(\alpha_1^j, \beta_t^j)$ and quite far. Finally, we add a thin rectangular pocket $\mathcal{P}_{j,r}$ to the left of the points $\alpha_1^j, \ldots, \alpha_t^j$ such that the uppermost longer side of the rectangular pocket lies on the line $\ell(\alpha_1^j, \alpha_t^j)$ (see Figure 2). We denote by $\mathcal{P}_{j,\alpha,\beta}$ the set of pockets $\{\mathcal{P}(c_1^j), \ldots, \mathcal{P}(c_t^j), \mathcal{P}(d_1^j), \ldots, \mathcal{P}(d_t^j)\}$ and call it *weak linker*.

If one wants to guard $\mathcal{P}_{j,\alpha,\beta}$ with only two points and place the first guard on α_i^j, one is not forced to place the second guard on β_i^j, as we would desire, but anywhere on an area whose uppermost point is β_i^j (see the shaded areas below the b_i^j's in Figure 2).

Linkers. For each $j \in [k]$, we allocate t points $\overline{\alpha}_1^j, \ldots, \overline{\alpha}_t^j$ on a horizontal line above and to the right of β_t^j at a quite large distance. We add two weak linkers $\mathcal{P}_{j,\alpha,\overline{\alpha}}$ and $\mathcal{P}_{j,\overline{\alpha},\beta}$, one linking $\alpha_1^j, \ldots, \alpha_t^j$ and $\overline{\alpha}_1^j, \ldots, \overline{\alpha}_t^j$, the other linking $\overline{\alpha}_1^j, \ldots, \overline{\alpha}_t^j$ and $\beta_1^j, \ldots, \beta_t^j$ (see Figure 3). We also add a thin horizontal pocket whose lowermost side is in the same line as the points $\overline{\alpha}_1^j, \ldots, \overline{\alpha}_t^j$. Pockets of $\mathcal{P}_{j,\alpha,\overline{\alpha}}$ and the two thin rectangular pockets force to put guards on α_i^j and $\overline{\alpha}_i^j$ (for a same $i \in t$), if we have only two guards to spare. Now, pockets of $\mathcal{P}_{j,\alpha,\beta}$ forces to place the third guard below β_i^j while pockets of $\mathcal{P}_{j,\overline{\alpha},\beta}$ forces to place the third guard above β_i^j (again if we have only three guards to spare). So, the only solution is to place the third guard exactly on β_i^j. The k linkers are placed accordingly to Figure 4.

Lemma 4 *$\forall j \in [k]$, $\forall i \in [t]$, the three associate points $\alpha_{i_1}^j$, $\overline{\alpha}_{i_2}^j$, $\beta_{i_3}^j$ guard entirely $\mathcal{P}_j$ iff $i_1 = i_2 = i_3$.*

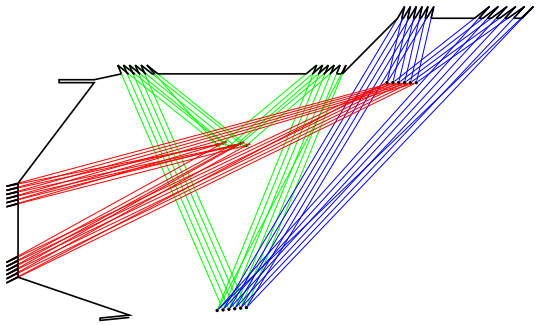


Figure 3: Point linker gadget: a triangle of (three) weak point linkers.

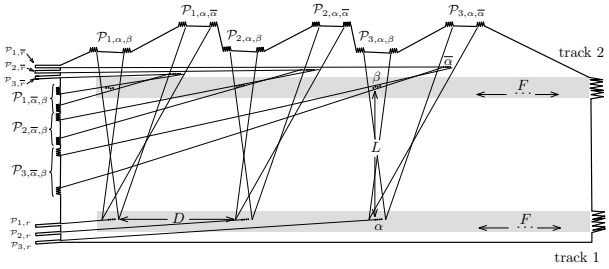


Figure 4: The overall picture of the reduction with $k = 3$.

4 Vertex Guard

The reduction is again from Structured 2-Track Hitting Set.

Vertex linkers. For each $j \in [k]$, permutation σ_j is encoded by a sub-polygon $\mathcal{P}_j$ that we call *vertex linker*, or simply *linker* (see Figure 5). We regularly set t consecutive vertices $\alpha_1^j, \alpha_2^j, \ldots, \alpha_t^j$ in this order, along the x-axis. Opposite to this *segment*, we place t vertices $\beta_{\sigma_j(1)}^j, \beta_{\sigma_j(2)}^j, \ldots, \beta_{\sigma_j(t)}^j$ in this order, along the x-axis, too. The $\beta_{\sigma_j(1)}^j, \ldots, \beta_{\sigma_j(t)}^j$, contrary to $\alpha_1^j, \ldots, \alpha_t^j$, are *not* consecutive. We put reflex vertices in between the vertices $\beta_{\sigma_j(1)}^j, \ldots, \beta_{\sigma_j(t)}^j$ to ensure that the only way of seeing entirely the *walls* $d^j e^j$ and $x^j y^j$ by taking two vertices α_i^j and $\beta_{i'}^j$ is that $i = i'$.

Lemma 5 *For any $j \in [k]$, the sub-polygon $\mathcal{P}_j$ is seen entirely by $\{\alpha_v^j, \beta_w^j\}$ iff $v = w$.*

What we should now prevent is that one puts a guard in a reflex vertex of the linker.

Filter gadget. The only way to see all the pockets of the filter gadget $\mathcal{F}_j$ (see Figure 6) with two guards is to place them on c_i and d_i for the same i. In the overall construction, the c_i's are in fact vertices $\beta_{\sigma_j(1)}^j, \ldots, \beta_{\sigma_j(t)}^j$. Thus, if one wants to guard

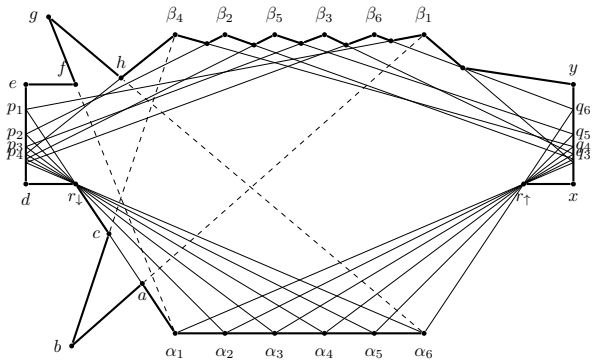


Figure 5: Vertex linker gadget. We omitted the superscript j in all the labels. Here, $\sigma_j(1) = 4,\ \sigma_j(2) = 2,\ \sigma_j(3) = 5,\ \sigma_j(4) = 3,\ \sigma_j(5) = 6,\ \sigma_j(6) = 1$.

all the pockets of $\mathcal{F}_j$ and $\mathcal{P}_j$ with only three guards, one should place them at vertices α_i^j, β_i^j, and $d^j_{\sigma_j(i)}$.

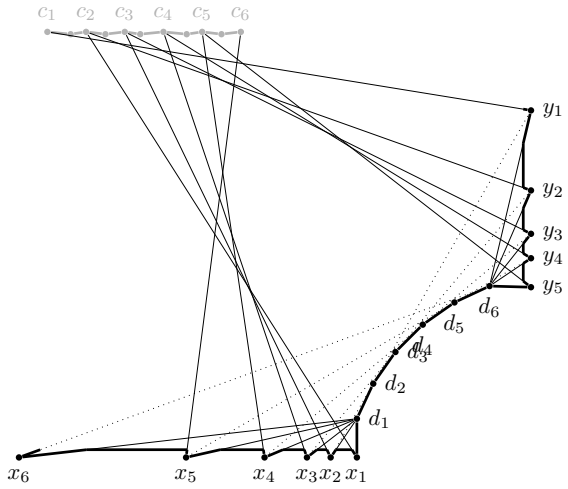


Figure 6: The filter gadget $\mathcal{F}_j$.

Overall construction. Permutation σ is encoded in the way depicted on Figure 7 by limiting the visibility of the vertices $\beta^j_{\sigma_j(1)}, \ldots, \beta^j_{\sigma_j(t)}$ to only one filter gadget, namely $\mathcal{F}_j$. Finally, as for the point guard variant, for each A- and B-interval, we place a triangular pocket seeing the corresponding vertices (see Track 1 and 2 of Figure 7).

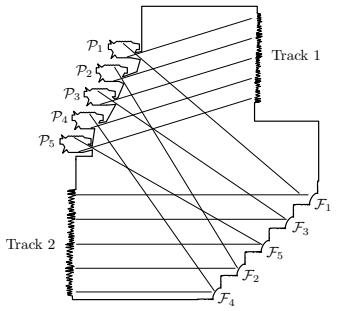


Figure 7: Overall picture of the reduction with $k = 5$.

Acknowledgments

Both authors are supported by the ERC grant PARAMTIGHT: "Parameterized complexity and the search for tight complexity results", no. 280152. We thank Meirav Zehavi and Saeed Merhabi for useful discussions during a preliminary stage of the project.

References

[1] S. Basu, R. Pollack, and M.-F. Roy. Algorithms in real algebraic geometry. *AMC*, 10:12, 2011.

[2] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[3] A. Deshpande. A pseudo-polynomial time $O(\log^2 n)$-approximation algorithm for art gallery problems. Master's thesis, Department of Mechanical Engineering, Department of Electrical Engineering and Computer Science, MIT, 2006.

[4] A. Deshpande, T. Kim, E. D. Demaine, and S. E. Sarma. A pseudopolynomial time $O(\log n)$-approximation algorithm for art gallery problems. In *WADS 2007*, pages 163–174, 2007.

[5] S. Durocher and S. Mehrabi. Guarding orthogonal art galleries using sliding cameras: algorithmic and hardness results. In *MFCS 2013*, pages 314–324. Springer, 2013.

[6] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

[7] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[8] S. Fisk. A short proof of chvátal's watchman theorem. *J. Comb. Theory, Ser. B*, 24(3):374, 1978.

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

[10] R. Impagliazzo and R. Paturi. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240. IEEE, 1999.

[11] R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *J. Comput. Syst. Sci.*, 40(1):19–48, 1990.

[12] P. Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104(1):1–16, 1998.

Visibility Testing and Counting for Uncertain Segments

Mohammad Ali Abam* Sharareh Alipour* Mohammad Ghodsi* † Mohammad Mahdian ‡

Abstract

We study two well-known planar visibility problems, namely visibility testing and visibility counting, in a model where there is uncertainty about the input data. The standard versions of these problems are defined as follows: we are given a set $\mathcal{S}$ of n segments in $\mathbb{R}^2$, and we would like to preprocess $\mathcal{S}$ so that we can quickly answer queries of the form: is the given query segment $s \in \mathcal{S}$ visible from the given query point $q \in \mathbb{R}^2$ (for visibility testing) and how many segments in $\mathcal{S}$ are visible from the given query point $q \in \mathbb{R}^2$ (for visibility counting).

In our model of uncertainty, each segment may or may not exist, and if it does, it is located in one of finitely many possible locations, given by a discrete probability distribution. In this setting, the probabilistic visibility testing problem (PVTP, for short) is to compute the probability that a given segment $s \in \mathcal{S}$ is visible from a given query point q and the probabilistic visibility counting problem (PVCP, for short) is to compute the expected number of segments in $\mathcal{S}$ that are visible from a query point q. We first show that PVTP is $\#P$-complete. In the special case where uncertainty is only about whether segments exist and not about their location, we show that the problem is solvable in $O(n \log n)$ time. Using this, together with a few old tricks, we can show that one can preprocess $\mathcal{S}$ in $O(n^5 \log n)$ time into a data structure of size $O(n^4)$ so that PVTP queries can be answered in $O(\log n)$ time. Our algorithm for PVTP combined with linearity of expectation gives an $O(n^2 \log n)$ time algorithm for PVCP. We also give a faster 2-approximation algorithm for this problem.

1 Introduction

Background. Visibility testing and visibility counting are basic problems in computational geometry. Visibility plays an important role in robotics and computer graphics. In robotics, for example, the efficient exploration of an unknown environment requires computing the visibility polygon of the robot or the number of visible objects from the robot or test whether the robot sees a specific object. In some computer graphics applications, also, it is important to identify the objects in a scene that are illuminated by a light source.

Two points $p, q \in \mathbb{R}^2$ are visible from each other with respect to $\mathcal{S}$, if there exists no segment $s \in \mathcal{S}$ intersecting line segment $\overline{pq}$. We say that a segment $\overline{st} \in \mathcal{S}$ is visible from a point p, if a point $q \in \overline{st}$ can be found from which p is visible. In this paper, we consider two planar visibility problems; visibility testing and visibility counting. For a set $\mathcal{S}$ of n segments in $\mathbb{R}^2$ and a point q, in visibility testing problem, we want to test whether q sees a given segment $s \in \mathcal{S}$. In visibility counting problem we want to count the number of segments in $\mathcal{S}$ that are visible from q. For simplicity we assume all the segments are contained in a bounding box.

Uncertain data. It is not surprising that in many real-world applications we face uncertainty about the data. For geometric problems like visibility, this means uncertainty about the location of the input set. There are multiple ways to model such uncertainty. For example, we can assume each object lies inside some region, but not exactly where in that region, and use this assumption to prove bounds on the quantity of interest. Such a model is used in [8]. Alternatively, we can use a discrete probability distribution to model uncertainty. This "stochastic" approach is used in [1, 10]. We choose the latter approach in this paper. In particular, our model of uncertainty is very similar to the model used in [10].

Related work. There is significant prior work on the non-stochastic version of the problems studied in this paper. There are some works dedicated not only to the exact computing [5, 11, 13] of the problem but also to approximate computing [3, 4, 7, 11]. In both, time-space trade-offs haven been considered.

In real application there are situations where we need to model the problems based on uncertain data (See [1, 8, 9]). In [6], they compute visibility between imprecise points among obstacles. This leads us to define the uncertain model of VTP and VCP and propose algorithms to solve them.

Problem statement. Suppose we are given a set $\mathcal{S}$ of n uncertain segments. More precisely, we are given a discrete probability distribution for each $s_i \in \mathcal{S}$, that is, we have a set $\mathcal{D}_i = \{s_{i,1}, \cdots, s_{i,m_i}\} \cup \{s_{i,0} = \perp\}$

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

†Institute for Research in Fundamental Sciences (IPM) School of Computer Science

‡Google research

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

of possible locations with associated probabilities $p_{i,j}$ such that $\mathbf{Pr}(s_i = s_{i,j}) = p_{i,j}$ and $\sum_j p_{i,j} = 1$. The special segment $\perp$ indicates that the segment s_i does not exist in $\mathcal{S}$. In this setting, the set $\mathcal{S}$ can be seen as a random variable (or random set) as it consists of probabilistic segments. This random variable gets its value from a sample space of size $\Pi_i(m_i+1)$ with the probability being equal to $\Pi_{s\in\mathcal{S}}\mathbf{Pr}(s)\Pi_{s\notin\mathcal{S}}(1-\mathbf{Pr}(s))$. To this end, assume $z = \max\{1+m_i\}$, i.e., z denotes the maximum size of the given distributions. A special case that we will pay special attention to is when $z = 2$. This is the case where the uncertainty is only about the existence of the segments, and not about their location.

It is natural to define the probabilistic version of visibility testing and visibility counting problems in the above setting where $\mathcal{S}$ is a random set:

- Probabilistic Visibility Testing Problem (PVTP): compute the probability that a given segment $s \in \mathcal{S}$ is visible from a given query point q.
- Probabilistic Visibility Counting Problem (PVCP): compute the expected number of segments in $\mathcal{S}$ being visible from q.

Our results. We first show that PVTP is $\#P$-complete. We then turn our attention to the special case where $z = 2$. We present an algorithm running $O(n \log n)$ time that answers PVTP. Then, we present a simple way of putting n uncertain segments into a data structure of size $O(n^4)$ such that queries can be answered in $O(\log n)$ time. Finally, we donate our attention to PVCP whose complexity class is unknown to us. Here, we present a polynomial-time 2-approximation algorithm that approximately solves PVCP. We then show how to preprocess $\mathcal{S}$ into a data structure of size $O(n^4)$ in order to approximately answer each query in $O(\log n)$ time.

2 Probabilistic visibility testing

We start by a simple polynomial-time reduction from #perfect-matching problem to PVTP in order to show PVTP is $\#P$-complete. The #perfect-matching problem of computing the number of perfect matching in a given bipartite graph, is known to be $\#P$-complete [12]. We next explain the details.

Suppose a bipartite graph $G = (U, V, E)$ is input to #perfect-matching problem where $U = \{u_1, \cdots, u_n\}$ and $V = \{v_1, \cdots, v_n\}$ are vertex parts of G and E is the edge set of G. For the given bipartite graph, we construct an instance of PVTP and introduce a query point q and a query segment s such that each perfect matching uniquely corresponds to one element of the sample space of uncertain segments in which s is not visible from q. Consider n intervals $[i, i+1]$ on the x-axis where i changes from 0 to $n-1$. Imagine the interval $[i, i+1]$ corresponds to the vertex v_i; denoted by $I(v_i)$. For each vertex $u_i \in U$, we define an uncertain segment $\mathcal{D}_i = \{I(v_j) | \{u_i, v_j\} \in E\}$ with the uniform distribution—note that in this instance each uncertain segment always exist. We add one more uncertain segment s consisting of one segment with probability 1 whose endpoints are $(0, -1)$ and $(n, -1)$. To this end, consider the query point q is anywhere above the x-axis with x-coordinate greater than 0 and less than n.

Segment s is not visible from q iff the interval $[0, n]$ is completely covered by the uncertain segments defined on the x-axis. There are n such uncertain segments and each covers exactly 1 unit of $[0, n]$. Therefore, each uncertain segment must cover exactly one of n unit intervals. This is the intuition behind one-to-one correspondence between perfect matching and the subset of the sample space in which s is not visible from q. Therefore, we conclude the following theorem.

Theorem 1 *PVTP is $\#P$-complete.*

From now on, we restrict ourself to the special case where $z = 2$, i.e., each uncertain segment either does not exist or exists in only one possible location. Suppose we are given n uncertain segments $s_1, \cdots, s_n$. Let $\mathbf{Pr}(s_i \in \mathcal{S}) = p_i$ which of course implies $\mathbf{Pr}(s_i \notin \mathcal{S}) = 1 - p_i$.

Next, we explain how to compute $\mathbf{Pr}(q \text{ sees } s)$ for the given segment s and point q. If $s \notin \mathcal{S}$, q of course can not see s. Therefore, $\mathbf{Pr}(q \text{ sees } s) = \mathbf{Pr}(q \text{ sees } s | s \in \mathcal{S})\mathbf{Pr}(s \in \mathcal{S})$. This reduces our task to computing of $\mathbf{Pr}(q \text{ sees } s | s \in \mathcal{S})$. Let Δ be a triangle with vertex q and side s. Every other uncertain segment that does not intersect Δ, can not prevent q to see s. Therefore, we can restrict ourself to uncertain segments intersecting Δ. We project these uncertain segments to s with respect to q. Now, as the main ingredient, we must solve the following problem:

- Suppose we are given n uncertain intervals $I = \{I_1, \cdots, I_n\}$ on the real line; each I_i exists with probability p_i. Compute the probability that the given interval $[a, b]$ is covered by the uncertain intervals, denoted by $\mathbf{Pr}([a, b] \text{ is covered})$.

Computing the desired probability seems needs $\Theta(2^n)$ time as the size of the sample space can be $\Theta(2^n)$ in the worst case. But, we next show how the dynamic paradigm helps us to perform the computation in $O(n \log n)$ time. For simplicity, we can assume the intervals have been sorted by their right endpoints and intersection of each I_i with $[a, b]$ is not empty. Let $r(I_i)$ ($l(I_i)$) be the right (left) endpoint of I_i. We present the following recursive formula.

For each point $a' \in [a, b]$, let $sol(a')$ be the probability that $[a', b]$ is covered. So, $sol(a)$ is the probability

that $[a, b]$ is covered. Let $S(a') = \{I'_1, ..., I'_l\}$ be the set of intervals that cover a' and they are sorted according to their right endpoints.

Lemma 2 *We define $sol(b) = 1$, then we have*

$$sol(a') = \sum_{j=1}^{l} p'_j(\prod_{i=1}^{j-1}(1 - p'_i))sol(r(I'_j)).$$

Proof. Suppose that $a' \in [a, b]$, so if $[a', b]$ is covered, then at least one of the segments in $S(a')$ should be chosen. There are l segments that cover a'. Since the segments in $S(a')$ are sorted according to their right endpoints then, the probability that I'_j is the first segment that covers a' is $p'_j \prod_{i'=1}^{j-1}(1 - p'_i)$. Recursively $[a', b]$ is covered with the probability of $sol(r(I'_j))$. So, we have $sol(a') = \sum_{j=1}^{l} p'_j(\prod_{i'=1}^{j-1}(1 - p'_i))sol(r(I'_j))$. □

Each right endpoint of the intervals can be covered by $O(n)$ of the intervals. In the recursive formula, we call each right endpoint at most once. For each $sol(r(I'_j))$ we have to compute $\prod_{i'=1}^{j-1}(1-p'_i)$, since the segments are sorted according to their right endpoint, for each $sol(r(I'_j))$ we multiply $\prod_{i'=1}^{j-2}(1-p'_i)$(the value of previous step) by $1-p'_j$, which means we can compute $sol(a)$ in $O(n^2)$ time. Next we propose a faster algorithm.

To fill the array sol, we sweep the endpoints from right to left and keep the track of all intervals intersecting the sweep line in a binary search tree (BST, for short) over the right endpoint of intervals supporting insertion/deletion in $O(\log n)$ time. We augment each node of the BST with extra values in order to expedite our computation as we explain next.

Upon processing a right end-point, say $r(I_i)$, we compute $sol(r(I_i))$, which is the sum of all the nodes of tree. This can be computed in $O(\log n)$ time. Then, we implicitly multiply all the nodes by $(1 - p_i)$ and then add $r(I_i)$ to the tree with the value of $p_i sol(r(I_i))$. For the left endpoint of an interval, $l(I_i)$, we delete I_i, from the tree and implicitly divide all the right endpoints greater than $r(I_i)$ by $(1 - p_i)$. This also can be done in $O(\log n)$ time. There are $O(n)$ endpoints, so the running time is $O(n \log n)$.

Theorem 3 *Given a point and a segment, PVTP can be answered in $O(n \log n)$ time when $z = 2$.*

Now, we preprocess the segments such that for any given query point q, PVTP can be answered in $O(\log n)$ time. First, connect each pair of the endpoints by a line and extend it until it hits the bounding box. These lines will partition the bounding box into $O(n^4)$ regions. For a fixed segment $s \in \mathcal{S}$, the answer to PVTP for all the points in a given region is the same, because the combinatorial order of segments that cover s is the same for all the points inside that region. Therefore, in the preprocessing time we choose a point q_i from each region r_i and compute $\mathbf{Pr}(q_i \text{ sees } s)$ in $O(n \log n)$ time. So, for a given set of segments $\mathcal{S}$ and a segment $s \in \mathcal{S}$, we preprocess the segments in $O(n^5 \log n)$ time and $O(n^4)$ space such that for any given query point q, we locate the region r_i containing q in $O(\log n)$ time and return $\mathbf{Pr}(q_i \text{ sees } s) = \mathbf{Pr}(q \text{ sees } s)$.

3 Probabilistic visibility counting

In this section we study the probabilistic visibility counting problem. We start with a few notations. For each subset $T \subset \mathcal{S}$, let $m_q(T)$ be the number of segments visible from q when the set of segments is T. So, the expected number of segments visible from q can be written as: $E(m_q) = \sum_{T \subseteq \mathcal{S}} \mathbf{Pr}(T) m_q(T)$, where $\mathbf{Pr}(T)$ denotes the probability that the set of realized segments is T.

Another way to compute $E(m_q)$ is using linearity of expectations: $E(m_q) = \sum_{i=1}^{n} \mathbf{Pr}(q \text{ sees } s_i)$.

For the case $z = 2$, we can use the above identity and the algorithm in the previous section to compute $E(m_q)$ in $O(n^2 \log n)$ time with no preprocessing. Also as in the previous section, we can use preprocessing to reduce query time: the answer of $PVCP$ is the same for all the points in each region in the space partition. So, we can compute this number for all the regions in $O(n^6 \log n)$ preprocessing time and $O(n^4)$ space, such that for any query point q, $E(m_q)$ can be answered in $O(\log n)$ time. Now, we show how to approximately solve this problem more efficiently.

3.1 Approximation of PVCP

In this section we propose a 2-approximation solution for PVCP. First, we present a theorem (the preliminary version of this theorem was published in [4]):

Theorem 4 *Let S be a set of disjoint line segments in the plane and $ve(q)$ be the number of visible endpoints of the segments and m_q be the number of visible segments, then we have*

$$m_q \leq ve(q) \leq 2m_q$$

Proof. We define four types of visible segments.

- R: the visible segments that their right endpoints are visible to q and their left endpoints are not visible to q.
- L: the visible segments that their left endpoints are visible to q and their right endpoints are not visible to q.
- LR: the visible segments that their right and left endpoints are visible to q.
- Mid: the visible segments that their right endpoints and left endpoints are not visible to q.

So, we have, $m_q = R + L + LR + Mid$. Note that for each segment s_i of type Mid or R, there should be a segment s_j of type R or LR, such that the ray emanating from q to the right endpoint of s_j cross s_i after crossing the right endpoint of s_j. This means $Mid + R \leq R + LR$. So, we have

$$m_q = R + L + LR + Mid \leq R + L + 2LR \leq 2(R + L + LR + Mid) = 2m_q \text{ or } m_q \leq ve_q \leq 2m_q.$$

□

Now, we use Theorem 4 to approximate PVCP. Let $m_q(T)$ and $ve_q(T)$ be the number of visible segments and visible endpoints, respectively in $T \subset \mathcal{S}$ w.r.t T, so we have $m_q(T) \leq ve_q(T) \leq 2m_q(T)$. So, we can conclude that,

$$\sum_{T\subset\mathcal{S}} \mathbf{Pr}(\mathcal{S} = T)m_q(T) \leq \sum_{T\subset\mathcal{S}} \mathbf{Pr}(\mathcal{S} = T)ve_q(T) \leq \sum_{T\subset\mathcal{S}} \mathbf{Pr}(\mathcal{S} = T)2m_q(T).$$

Or in other words,

$$E(m_q) \leq E(ve_q) \leq 2E(m_q).$$

So, we compute $E(ve_q) = \sum_{i=1}^{n} \mathbf{Pr}(r(s_i) \text{ sees } q) + \mathbf{Pr}(l(s_i) \text{ sees } q)$.

$$\mathbf{Pr}(r(s_i) \text{ sees } q)) = \sum_{j=1}^{z} p_{i,j}\mathbf{Pr}(r(s_{i,j}) \text{ sees } q).$$

Let $s_{k,1'}, s_{k,2'}, ..., s_{k,l'}$ be the possible locations of s_k in $\mathcal{D}_k$ that cross $\overline{r(s_{i,j})q}$, the probability that s_k does not intersect $\overline{r(s_{i,j})q}$ is $p_k^{i,j} = (1 - p_{k,1'} - p_{k,2'} - ... - p_{k,l'})$.

$$\mathbf{Pr}(q \text{ sees } r(s_i)) = \sum_{j=1}^{z} p_{i,j}p_1^{i,j}.p_2^{i,j}....p_n^{i,j}$$

We have $2nz$ possible locations for the endpoints and we can compute $P(q \text{ sees } r(s_i))$ in $O(zn)$, so $E(ve(q))$ is computed in $O(n^2z^2)$.

For $z = 2$ we present a faster algorithm. Suppose that $a \in s_i$ is an endpoint of s_i. Let $s'_1, s'_2, ..., s'_k$ be the set of segments that intersect $\overline{aq}$, since the probability of selection of the segments are independent, we have

$$\mathbf{Pr}(q \text{ sees } a) = p_i.(1 - p'_1)(1 - p'_2)...(1 - p'_k).$$

Which yields: $E(ve_p) = \sum_{a\in s_i} \mathbf{Pr}(q \text{ sees } a)$.

So, for each endpoint, we need the segments that intersect $\overline{aq}$. We use the following theorem:

Theorem 5 [2] *Let S be a set of n segments in the plane and $n \leq k \leq n^2$, we can preprocess the segments in $O_\epsilon(k)$ such that for a given query segment s, the number of segments crossed by s can be computed in $O_\epsilon(n/\sqrt{k})$.*

By Theorem 5 we can compute $\mathbf{Pr}(q \text{ sees } a)$ in $O(n/\sqrt{k})$. So, for $2n$ endpoints, $E(ve_p)$ is computed in $n.O(n/\sqrt{k})$. If $k = n^{\frac{4}{3}}$, then we have:

Theorem 6 *Let, S be a set of given segments and q be a given point. If each segment is chosen with probability p_i, then, the expected number of visible endpoints from q can be computed in $O(n^{\frac{4}{3}})$ which is a 2-approximation of $E(m_q)$.*

Acknowledgments

We thank Mahdi Safarnejad for his comments and helps.

References

[1] Abam, M. A., De Berg, M., and Khosravi Piecewise-linear approximations of uncertain functions *Algorithms and Data Structures*, 1–12, 2011.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, Advances in Discrete and Compu- tational Geometry, volume 223 of Contemporary Mathematics, pages 156. American Mathematical Society Press, 1999.

[3] Alipour, S., Ghodsi, M., Zarei, A., and Pourreza, M. Visibility testing and counting. *Information Processing Letters*, 115(9), 649-654, 2015.

[4] Alipour, S., Zarei, A. Visibility Testing and Counting. *FAW-AAIM 2011, Jinhua, China, LNCS*, (Volume 6681) by Springer-Verlag: 343-351, 2011.

[5] Asano, T. An efficient algorithm for finding the visibility polygon for a polygonal region with holes. *IEICE Transactions*, 557–589, 1985.

[6] Buchin, K., Kostitsyna, I., Loffler, M., and Silveira, R. I. Region-based approximation of probability distributions (for visibility between imprecise points among obstacles) *CoRR, abs/1402.5681*, 2014

[7] Gudmundsson, J., Morin, P. Planar visibility: testing and counting. *Annual Symposium on Computational Geometry*, 77–86, 2010.

[8] van Kreveld, M., and Loffler, M. Approximating largest convex hulls for imprecise points. *Journal of Discrete Algorithms*, 6(4), 583-594, 2008.

[9] Loffler, M., and van Kreveld, M. Largest bounding box, smallest diameter, and related problems on imprecise points. *In Algorithms and Data Structures, Springer Berlin Heidelberg.*, 446–457, 2007.

[10] Munteanu, A., Sohler, C., and Feldman, D. Smallest enclosing ball for probabilistic data. *Proceedings of the thirtieth annual symposium on Computational geometry. ACM*, 2014.

[11] Suri, S. and O'Rourke, J. Worst-case optimal algorithms for constructing visibility polygons with holes. *In Proceedings of the Second Annual Symposium on Computational Geometry (SCG 84)*, 14–23, 1984.

[12] Valiant, L. G. The complexity of computing the permanent. *Theoretical computer science*8(2):189–201 ,1979.

[13] Vegter, G. The visibility diagram: A data structure for visibility problems and motion planning. *In: Gilbert, J.R., Karlsson, R. (eds.) SWAT(1990). LNCS,* 447:97–110. Springer, Heidelberg, 1990.

Covering Polygons with Rectangles

Roland Glück*

Abstract

A well-known and well-investigated family of hard optimization problems concerns variants of the cutting stock or nesting problem, i.e. the non-overlapping placing of polygons to be cut from a rectangle or the plane whilst minimizing the waste. Here we consider an in some sense inverse problem. Concretly, given a set of polygons in the plane, we seek the minimum number of rectangles of a given shape such that every polygon is covered by at least one rectangle. As motions of the given rectangle we investigate the cases of translation and of translation combined with rotation.

1 Introduction

In manufacturing, one often faces the problem of cutting a set of given polygons out of a piece of material (e.g. sheet metal or cloth) in a way which produces as less waste as possible. In this paper we investigate the subsequent step in production technology: once the pieces are cut out they will be picked off and transported by a suitable device. Here we restrict ourselves to a rectangular gripper and various degrees of freedom: The first case is a rectangular gripper which can be translated both in x- and y-direction; the second case deals with a rectangular gripper which additionally has the possibility of being rotated. The concrete motivation of this paper is a machine which cuts polygons out of a carbon fiber fabric and grasps the cut pieces with a rectangular gipper with vacuum suction devices.

Basically, this task corresponds to covering a set of polygons by copies of a rectangle such that every polygon is contained in at least one rectangle. There is a lot of work about covering sets of points with rectangles as in [4, 6] but none of them matches our problem. Due to the NP-hardness of all these problems (see [5] for a comprehensive list) we suspect that the problems we consider are also NP-hard. We do not propose an approximation algorithm but a family of exact algorithms which works well on practical instances.

The paper is organized as follows: Section 2 provides definitions and states the problem in a generic way. In Section 3 we prove some useful lemmata for the further course. A generic approach to the problems is presented in Section 4, while Section 5 deals with some implementation issues and provides experimental results. The finishing Section 6 gives a short summary and directions of future work.

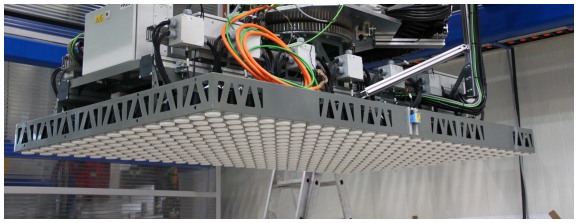


Figure 1: The bottom side of the gripper

2 Definitions

In order to formalize our task we introduce the concept of a packing: a *packing* $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$ is a set of n possibly overlapping simple polygons P_1, P_2, …, P_n. Clearly, $|\mathbf{P}|$ denotes the number of polygons of $\mathbf{P}$, and we use the notation $\|\mathbf{P}\|$ for the overall number of vertices in $\mathbf{P}$. We say that a packing $\mathbf{P}$ is *covered* by a set $\mathbf{C} = \{R_1, R_2, \ldots, R_m\}$ of rectangles if each polygon of $\mathbf{P}$ is contained in at least one rectangle of $\mathbf{C}$. If a rectangle R' arises from a rectangle R by a translation we call R' a *translation* of R, and if R' arises from R by a translation and a rotation we say that R' is a *general motion* of R (this is equivalent to the term "rigid motion" in [2]). With these namings we can define the main theme of our investigations:

Definition 1 *Let* $\mathbf{P}$ *be a packing and* R *an axis-aligned rectangle (the so-called gripper). We call a set of rectangles* $\mathbf{C} = \{R_1, R_2, \ldots, R_m\}$ *a* translational (general) cover *of* $\mathbf{P}$ *if* $\mathbf{C}$ *covers* $\mathbf{P}$ *and all rectangles of* $\mathbf{C}$ *are translations (general motions) of* R*.*

Since we are interested in covering a packing with as few as possible rectangles we call a cover of every kind *optimal* if it has minimal cardinality amongst all covers of the respective kind. To ease wording we refer by the term *cover* to both a translational or general cover. For a packing $\mathbf{P}$ and a rectangle R we denote the set of polygons of $\mathbf{P}$ covered by R by $\mathsf{cov}(R, \mathbf{P})$. We extend this notion to a set $\mathbf{R}$ of rectangles by $\mathsf{cov}(\mathbf{R}, \mathbf{P}) := \bigcup_{R \in \mathbf{R}} \mathsf{cov}(R, \mathbf{P})$.

*German Aerospace Center roland.glueck@dlr.de

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

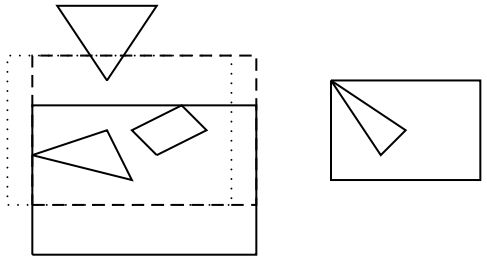


Figure 2: Translational Alignments

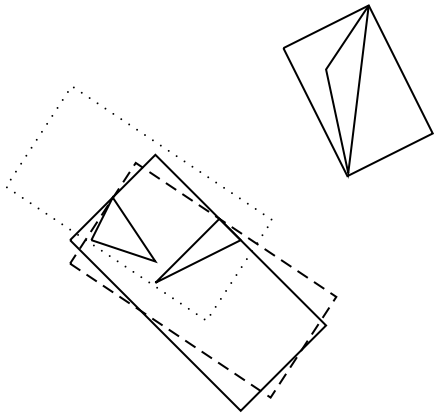


Figure 3: General Motion Alignments

3 Basic Facts and Observations

In general, the set of covers of a given packing is uncountable so we will have to discretize the search space in a suitable manner. As tools for reducing the number of coverings to consider for computation we state some useful properties and lemmata. The first lemma which holds for both translational and general covers will pave the way for a recursive approach to our problem:

Lemma 1 *Let* $\mathbf{P}$ *be a packing,* $\mathbf{C}$ *an optimal cover of* $\mathbf{P}$ *and* R_j *an arbitrary rectangle of* $\mathbf{C}$*. Then* $\mathbf{C}\backslash\{R_j\}$ *is an optimal cover of* $\mathbf{P}\backslash\mathsf{cov}(R_j, \mathbf{P})$*.*

Proof. Assume there is a cover $\mathbf{C}'$ of $\mathbf{P}\backslash\mathsf{cov}(R_j, \mathbf{P})$ with $|\mathbf{C}'| < |\mathbf{C}| - 1$. Then $\mathbf{C}' \cup \{R_j\}$ is a cover of $\mathbf{P}$ with a size of $|\mathbf{C}'| + 1 < |\mathbf{C}|$ which contradicts the optimality of $\mathbf{C}$. □

In the next lemma we give a first step towards discretization of the search space in the case of a translational cover:

Lemma 2 *Let* $\mathbf{P}$ *be a packing and* $\mathbf{C} = \{R_1, R_2, \ldots, R_m\}$ *an optimal translational cover of* $\mathbf{P}$*. Then there are points* p_1 *and* p_2 *of* $\mathbf{P}$ *and an index* j *together with an axis-aligned rectangle* R'_j *fulfilling the following properties:*

1. p_1 *has minimal x-coordinate amongst all points of* $\mathbf{P}$*,*
2. p_1 *lies on the left side of* R'_j*,*
3. p_2 *lies on the upper side of* R'_j*,*
4. *there are polygons* $P_1, P_2 \in \mathbf{P}$ *such that for* $i \in \{1, 2\}$ p_i *is a vertex of* P_i *and* P_i *is contained in* R'_j*, and*
5. $\mathbf{C}\backslash\{R_j\} \cup \{R'_j\}$ *is an optimal translational cover of* $\mathbf{P}$*.*

Note that we do not require that p_1 and p_2, P_1 and P_2 as well as R_j and R'_j are distinct. Moreover, if p_1 and p_2 are equal then they coincide with the upper left vertex of R'_j.

Proof. Let p_1 be a point of $\mathbf{P}$ with minimal x-coordinate and P_1 a polygon of $\mathbf{P}$ which has p_1 as a vertex. Then there is a rectangle $R_j \in \mathbf{C}$ containing P_1. Now we translate R_j in positive x-direction till p_1 lies on the left side of the translated rectangle $\hat{R}_j$. Clearly, we have $\mathsf{cov}(\hat{R}_j, \mathbf{P}) \supseteq \mathsf{cov}(R_j, \mathbf{P})$. Subsequently, we translate $\hat{R}_j$ in negative y-direction till a point p_2 with the following properties lies on the upper side of the translated rectangle R'_j:

1. All polygons of $\mathbf{P}$ with p_2 as a vertex contained in $\hat{R}_j$ are contained in R'_j, and
2. p_2 is a point with maximal y-coordinate fulfilling the above requirements.

Then we have $\mathsf{cov}(R'_j, \mathbf{P}) \supseteq \mathsf{cov}(\hat{R}_j, \mathbf{P}) \supseteq \mathsf{cov}(R_j, \mathbf{P})$, so $\mathbf{C}\backslash\{R_j\} \cup \{R'_j\}$ is indeed an optimal translational cover of $\mathbf{P}$. Moreover, p_1, p_2 and P_1 meet their requirements by construction, and for P_2 we can choose an arbitrary polygon with p_2 as a vertex which is contained in R'_j. □

The general situation is depicted in the left part of Figure 2: R_j correponds to the dotted rectangle, $\hat{R}_j$ to the dashed one, and the final rectangle R'_j is drawn with a full line. A pathological example where p_1 and p_2 as well as P_1 and P_2 coincide can be seen in the right part of the same figure.

A similar property can be stated for general covers (this and the previous lemma show some similarity to the term "stable placement" in[1]):

Lemma 3 *Let* $\mathbf{P}$ *be a packing and* $\mathbf{C}$ *an optimal general cover of* $\mathbf{P}$*. Then for every polygon* $P_{pi} \in \mathbf{P}$ *there are points* p_1*,* p_2 *and* p_3 *of* $\mathbf{P}$ *and an index* j *together with a rectangle* R'_j *fulfilling the following properties:*

1. *R'_j contains P_{pi},*
2. *p_1 and p_2 are distinct and lie on two different adjacent sides of R'_j*
3. *p_1, p_2 and p_3 lie on sides of R'_j,*
4. *there are polygons $P_1, P_2, P_3 \in \mathbf{P}$ such that for $i \in \{1,2,3\}$ p_i is a vertex of P_i and P_i is contained in R'_j,*
5. *$\mathbf{C}\backslash\{R_j\} \cup \{R'_j\}$ is an optimal general cover of $\mathbf{P}$.*

Proof. Let $R_j \in \mathbf{C}$ be a rectangle containing P_{pi}. We apply to R_j similar translations as in Lemma 2 but do not translate in x- and y-direction but in directions parallel to adjacent sides of R_j. Doing so, we end up with a rectangle $\hat{R}_j$ and two (not necessarily distinct!) points p_1 and p_2 with the following properties:

1. $\hat{R}_j$ contains P_{pi},
2. p_1 and p_2 lie on adjacent sides of $\hat{R}_j$,
3. $\mathsf{cov}(\hat{R}_j, \mathbf{P}) \supseteq \mathsf{cov}(R_j, \mathbf{P})$, and
4. there are polygons $P_1, P_2 \in \mathbf{P}$ such that for $i \in \{1,2\}$ p_i is a vertex of P_i and P_i is contained in $\hat{R}_j$.

Now we perform a general motion of $\hat{R}_j$ combined of a clockwise rotation and suitable translation which keeps p_1 and p_2 on their respective sides. There are two cases:

1. p_1 and p_2 coincide. Then the described general motion is a simple rotation of $\hat{R}_j$ around p_1. This rotation is continued as long as a point p_3 lies on a side of the resulting rectangle R'_j such that the following properties hold:
   (a) R'_j contains P_{pi},
   (b) there are polygons $P_1, P_3 \in \mathbf{P}$ such that for $i \in \{1,3\}$ p_i is a vertex of P_i and P_i is contained in R'_j, and
   (c) $\mathsf{cov}(R'_j, \mathbf{P}) \supseteq \mathsf{cov}(\hat{R}_j, \mathbf{P})$.
2. p_1 and p_2 are distinct. Here we continue the general motion till one of the following two cases concerning the arising rectangle R'_j occurs:
   (a) p_1 or p_2 coincide with a vertex of R'_j, or
   (b) there is a point on a side of R'_j such that
      i. R'_j contains P_{pi},
      ii. there are polygons $P_1, P_2, P_3 \in \mathbf{P}$ such that for $i \in \{1,2,3\}$ p_i is a vertex of P_i and P_i is contained in R'_j, and
      iii. $\mathsf{cov}(R'_j, \mathbf{P}) \supseteq \mathsf{cov}(\hat{R}_j, \mathbf{P})$.

Now, after possibly necessary renamings, p_1, p_2 and p_3 together with R'_j meet the requirements of the lemma. □

The general situation is shown in the lower left part of Figure 3: the dotted rectangle corresponds to R_j, the dashed one to $\hat{R}_j$ and the fully lined to R'_j. An extreme situation is illustrated by the upper right part of the same figure.

4 General Approach

We will now introduce an exact generic algorithm for the minimal cover problem. For the sequel we fix a rectangle R which we will use as gripper for a translational or general cover of a packing $\mathbf{P}$.

Let us assume we have an algorithm candidate_rectangles which determines for every packing $\mathbf{Q}$ a finite set of translations or general motions of R such that for every optimal cover $\mathbf{C}$ of $\mathbf{Q}$ and every $R_{cov} \in \mathbf{C}$ there is an $R_{cand} \in$ candidate_rectangles such that $\mathsf{cov}(R_{cov}, \mathbf{Q}) \subseteq \mathsf{cov}(R_{cand}, \mathbf{Q})$ holds. Together with a function simp_cov which computes an arbitrary cover (which can be done by packing each polygon into a rectangle of the gripper's shape) we can formulate the generic Algorithm 1 whose correctness is ensured by Lemma 1.

Algorithm 1 Generic Branch and Bound Algorithm

Require: A Packing $\mathbf{P}$ and a Gripper R
1: set<rectangle> *global_cover* = simp_cov($R, \mathbf{P}$)
2: int *global_depth* = |*global_cover*|
3: Branch and Bound(0,∅)
Ensure: *global_cover* is an optimal cover of $\mathbf{P}$ by R with cardinality *global_depth*

4: **function** Branch and Bound(int *depth*, set<rectangle> *rectangles*)
5: **if** *rectangles* covers $\mathbf{P}$ **then**
6: *global_depth* = *depth*
7: *global_cover* = *rectangles*
8: **else if** *depth* < *global_depth* **then**
9: set<rectangle> *candidate_rectangles* =
10: candidate_rectangles($\mathbf{P}\backslash$cov(*rectangles*, $\mathbf{P}$))
11: **for all** $R \in$ *candidate_rectangles* **do**
Branch and Bound(*depth* + 1, *rectangles* ∪ {R})
12: **end for**
13: **end if**
14: **end function**

Depending on whether one is interested in an optimal translational or general cover the function candidate_rectangles has to be implemented in different ways. Some possibilities are described in the next section.

Algorithm 1 is an exact algorithm so we cannot expect a polynomial running time. In the worst case, there are $\mathcal{O}(|\mathbf{P}|^{2|\mathbf{P}|})$ calls of Branch and Bound. As we will see in the next section, the candidate rectangles can be computed in $\mathcal{O}(\|\mathbf{P}\|)$ time for the translational and in $\mathcal{O}(\|\mathbf{P}\|)^3$ time for the general case.

5 Implementation Sketch and Experimental Results

A short look at Algorithm 1 reveals that a BFS in the induced search graph will lead to a faster implementation. As usual, the drawback of this approach is a greater amount of space required during the computation. We implemented both versions and observed that the BFS approach fits our practical problems better.

Lemmata 2 and 3 provide methods for computing the set *canditate_rectangles* in Line 9 of Algorithm 1.

In the case of a translational cover we observe that a translation of a rectangle is uniquely determined by the position of its upper left vertex. Moreover, given two distinct points, there at most one translations which make the two points lie on adjacent sides according to Lemma 2. In the sequel we will concentrate on the general cover problem because our gripper can also be rotated around the z-axis. Nevertheless, we implemented our algorithm for the translational cover and could solve instances with 25 polygons and 1250 vertices in less than a second on an Intel i7-4770 CPU with 3.4 GHz.

Similarly, each pair or triple of distinct point gives raise to only a finite number of general motions of given rectangle meeting the requirements from Lemma 3. So we iterate over the points or pairs or triples of points from the packing under consideration (concretely $\mathbf{P}\backslash\mathsf{cov}(\mathit{rectangles}, \mathbf{P})$ in Line 9 of Algorithm 1) and determine all motions of R which fulfill the conditions of Lemma 2 or 3, resp. Of course, it suffices to keep only those rectangles which cover a maximal set of polygons.

We refined this approach by the following idea: first, for every polygon P from the initial nesting, we generate a list $P_1^P, P_2^P, \ldots, P_m^P$ of compatible polygons which can be covered by the given gripper together with P. Second, we use these lists to compute the candidate rectangles for a packing $\mathbf{P'}$ arising during the execution as follows: we choose a pivot polygon P_{pi} from $\mathbf{P'}$ and iterate over all triples $(P_i^{P_{pi}}, P_j^{P_{pi}}, P_k^{P_{pi}})$ of compatible polygons of P_{pi} with $i \leq j \leq k$. For every such triple we compute the convex hull and determine for every tuple respectively triple of points of the convex hull the rectangles according to Lemma 3. As mentioned above, we only keep the rectangles covering a maximal set of polygons. The restriction to the points of the convex hull is justified by the fact that the covering rectangle has to contain all polygons of $\{P_i^{P_{pi}}, P_j^{P_{pi}}, P_k^{P_{pi}}\}$ which is equivalent to it that it contains the convex hull of these polygons. A crucial point is the choice of the polygon P_{pi} from Lemma 3 as pivot polygon in order to keep the branching degree of the algorithm at a low level. Experiments showed that a good choice for the pivot polygon is a polygon which has minimal distance to a vertex of an axis parallel rectangular minimal bounding box of $\mathbf{P'}$.

As one would expect, our experiments indicated a running time cubic in the overall number of vertices and roughly exponential in the number of polygons. Our implementation in Java solved instances from practice with 25 polygons and 1250 vertices in about 20 minutes on average in the same environment as above.

6 Conclusion and Outlook

Our algorithm seems to be applicable to practical instances. However, there is room for further improvements. In our setting the algorithm was run on only one core which is clearly not optimal since it is obviously easy to construct a parallelized version. Another idea is to compare other strategies than described above for finding the pivot polygon. From a theoretical point of view, it will be interesting to show the conjectured NP-completeness of our problems.

Acknowledgments

The author is grateful to Torben Hagerup, Christian Rähtz, Lev Sorokin and the anonymous reviewers for valuable hints and remarks.

References

[1] Chazelle, B.: The polygon containment problem. Advances in Computing Research, 1–33. JAI Press (1983)

[2] Dickerson, M., Scharstein, D.: Optimal placement of convex polygons to maximize point containment. Computationl Geometry 11(1), 1–16 (1998)

[3] Dowsland, K.A.: Determining an upper bound for a class of rectangular packing problems. Computers & OR 12(2), 201–205 (1985)

[4] Fowler, R.J., Paterson, M., Tanimoto, S.L.: Optimal packing and covering in the plane are np-complete. Inf. Process. Lett. 12(3), 133–137 (1981)

[5] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

[6] Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32(1), 130–136 (1985)

Two-Dimensional Closest Pair Algorithms in the VAT-Model

Fabian Dütsch*

Abstract

Recently, Jurkiewicz and Mehlhorn [10] observed that the cost of virtual address translation affects the practical runtime behavior of several fundamental algorithms on modern computers. We extend their results to two dimensions by analyzing and experimentally evaluating algorithms for the closest pair problem regarding the impact of address translation.

1 Introduction

Modern computer systems feature a multi-level memory hierarchy and virtual memory, which cannot be modeled adequately with the RAM-model. Usually, the number of cache misses in the presence of a memory hierarchy is examined in the EM- [1] and CO-model [6]. A machine modeled in these models consists of a slow main memory of unlimited size and a fast cache of size M. Data are moved between these levels in blocks of size B. In contrast to EM-algorithms, cache-oblivious algorithms assume the optimal cache replacement strategy and must not refer to the parameters M and B in the code.

However, these models do not cover the cost of virtual address translation. Running processes access their own linear address spaces via virtual addresses. The operating system transparently maps virtual to physical addresses, typically by walking a root-to-leaf path in a process-specific translation tree. Jurkiewicz and Mehlhorn observed impacts of the cost of virtual address translation on the practical runtime of several fundamental algorithms: the observed runtime of scanning an array in random order, binary searching, and sorting by heapsort exceeds the predicted RAM- and I/O-complexities by a factor of $\mathcal{O}(\log n)$ [10]. As these results cannot be explained by any of the former models, Jurkiewicz and Mehlhorn developed a new model of computation, that considers the cost of virtual address translation: the Virtual Address Translation (VAT) model. Their analyses show that the VAT-complexities match the measured runtime behaviors.

In this note, we revisit the two-dimensional closest pair problem in the VAT-model. We analyze and experimentally evaluate the algorithms by Bentley and Shamos [4], Hinrichs *et al.* [9] and Golin *et al.* [7].

*Department of Computer Science, Westfälische Wilhelms-Universität Münster, Germany, f.duetsch@uni-muenster.de

2 The Model

The *VAT-model* [10] extends the EM-model by modeling virtual addresses and their translation cost. A machine modeled in the VAT-model features two memory levels, both of which are partitioned into pages of size P (corresponding to B). The single processor cannot directly access the main memory, but first has to load the concerned page into the *translation cache* (TC) of the size of W pages (corresponding to M/B blocks). The cost of a cache fault is τ times the cost of a RAM-operation, where τ is some positive machine parameter.

The program accesses the main memory via virtual addresses. An address is a $(d+1)$-tuple in $\{0,\dots,K-1\}^d \times \{0,\dots,P-1\}$; the first part, called the *index*, determines the page and the second part the offset within the page. To translate a virtual address, one has to traverse a root-to-leaf path in a K-ary *translation tree* of height $d := \lceil \log_K(\text{max used page}) \rceil$. Leaves correspond to data pages, i.e., physical pages storing the data. The translation nodes of the *translation path* are determined by the components of the address index and are accessed successively. The internal nodes of size P as well as leaves are stored in the physical main memory and have to be present in the TC when being accessed.

The VAT-complexity is given by the number of cache faults incurred by accesses to data pages and translation nodes. Hence, it may exceed the I/O-complexity by a factor of $\mathcal{O}(d) = \mathcal{O}(\log_K(m/P))$ and the RAM-complexity by a factor of $\mathcal{O}(\tau d)$, with m being the program's memory consumption. The *asymptotic order relations* assume, among others, that following relations hold: [10]

(A1) $1 \le \tau d \le P$, i.e., loading a translation path can be amortized over P RAM-operations.

(A2) $d \le W < m^\theta$, for $\theta \in (0,1)$, i.e., the memory consumption is much bigger than the cache size.

Furthermore, we assume that the root of the translation tree is always present in the TC.

2.1 Previous results

To analyze cache-oblivious algorithms in the VAT-model, Jurkiewicz and Mehlhorn interpreted the translation tree as a $(d+1)$-level memory hierarchy: for $0 \le i \le d$, the translation nodes of height i correspond to blocks of size $K^i P$ and form a memory level.

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

By uniformly allocating the TC to these memory levels, the following statement results.

Theorem 1 ([10]) *A cache-oblivious algorithm with I/O-complexity $C(M, B, n)$, where M is the size of the cache and B is the size of a block, incurs maximally $\sum_{i=0}^{d-1} C(\lfloor \frac{W}{d} \rfloor K^i P, K^i P, n)$ cache faults in the VAT-model with optimal replacement strategy.*

Thus, linear scanning with I/O-complexity $1 + \lceil \frac{n}{B} \rceil$ causes at most $2d + \frac{K}{K-1}\frac{n}{P}$ faults in the VAT-model. However, this theorem cannot be applied to many cache-oblivious algorithms, as the tall-cache assumption $M \in \Omega(B^2)$ does not hold on each level of the corresponding memory hierarchy. To address this problem, Jurkiewicz *et al.* derived the upper bound $4d \cdot C(\frac{PW}{4}, dP, n)$ for a larger class of algorithms [11]. It implies that I/O-optimal sorting algorithms sort with at most $\mathcal{O}(\frac{n}{P}\lceil \frac{\log n/(PW)}{\log W/d} \rceil)$ faults in the VAT-model. For realistic cache sizes $W \in \Omega(d^2)$, this complexity equals the optimal I/O-complexity. Furthermore, taking into account $\frac{\tau}{P} \overset{(A1)}{\leq} \frac{1}{d}$ shows, that the VAT-cost of sorting is dominated by the RAM-cost $\mathcal{O}(n \log n)$.

3 Preliminaries

We start by analyzing several building blocks used in the closest pair algorithms in the VAT-model.

3.1 Search Data Structures

When searching for a random element, at least one random memory access is necessary. In this case, the best strategy is to cache the upper nodes of the translation tree. We can prove:

Proposition 2 *The average number of cache faults incurred by a search for a random and uniformly distributed element in a data structure storing n elements is at least $\lceil \log_K \frac{n}{P(W+1)} \rceil - 1$.*

This implies that the VAT-complexity of hashing, unlike its RAM- and I/O-complexity, is not constant. At the same time, the amortized number of faults caused by perfect hashing with constant amortized RAM-complexity does not exceed $\Theta(\log_K \frac{n}{PW})$.

The VAT-complexity of accessing and updating a search tree depends on its memory layout and the cache replacement strategy. A search in a balanced binary tree with a random layout incurs $\mathcal{O}(\log \frac{nd}{W} \log_K \frac{n}{PW}) \overset{(A2)}{=} \mathcal{O}(\log \frac{n}{W} \log_K \frac{n}{PW})$ cache faults, if the upper nodes of the search tree and of the translation tree are cached. Searching in a B-tree [2] causes $\mathcal{O}(\log_P \frac{n}{W} \log_K \frac{n}{PW})$ faults. The multi-level blocking regarding blocks of sizes $K^i P$, for $0 \leq i \leq d$, generates a static, cache-aware layout with search cost of $\sum_{i=0}^{d-1} \lceil \log_{K^i P} n \rceil \leq d + \log_P n + \log_K n \ln \log_P n$ cache faults, if searches start with an empty TC. Otherwise, the number of cache faults is $\mathcal{O}(\log_P \frac{n}{W} + \log_K \frac{n}{W} \log \log_P \frac{n}{W})$, if $W \geq 2d$. Applying Theorem 1 to the search in the cache-oblivious van Emde Boas layout [12] results in the same VAT-complexity [10]. Furthermore, the cache-oblivious layout can be dynamized to support searches and updates in the same worst-case complexity (see, e.g., [3]). As the following theorem shows, this is optimal.

Theorem 3 *The average- and worst-case complexities of the number of faults incurred by comparison-based searching is $\Theta(\log_P \frac{n}{W} + \log_K \frac{n}{W} \log \log_P \frac{n}{W})$.*

Proof Sketch. To establish a lower bound, we separately consider a problem with lower I/O-complexity $C(M, B, n)$ on the levels of the translation tree. Interpreting the levels as different levels of a memory hierarchy shows that $\sum_{i=0}^{d} C(WK^i P, K^i P, n)$ is a lower bound for the number of cache faults in the VAT-model. In both cases, plugging in the lower I/O-bound $\Omega(\log_B \frac{n}{M})$ of comparison-based searching results in the claimed lower VAT-complexity. □

The proof carries over to each problem for which an optimal cache-oblivious algorithm whose I/O-complexity does not depend on M is known. In this case, the stated lower bound matches the upper bound from Theorem 1.

In summary, it can be stated that the cost of virtual address translation usually dominates the RAM-complexity of searching.

3.2 Divide and Conquer

In the RAM-model, the complexity of divide-and-conquer algorithms is often determined using the master theorem [5]. However, it is not suitable for the VAT-model, as it may eliminate additional parameters, such as P and W. Additionally, it does not consider that cache faults can be prevented at deep recursive levels in certain cases. We thus adapt the master theorem to the VAT-model.

Theorem 4 *Let $a \in \mathbb{N}$ and $b \in \mathbb{R}$ be constants, with $b > 1$, and let $V \in \mathbb{N}$. Let C, f, and g be asymptotic positive functions. Consider an in-place, divide-and-conquer algorithm incurring $C(W, P, K, d, n) \leq a \cdot C(W, P, K, d, \lceil \frac{n}{b} \rceil) + g(W, P, K, d) \cdot f(n)$ faults. Then, the number of cache faults incurred by an asymptotically optimal replacement strategy on a TC of size $W \geq V \geq 2d + \Omega(d)$ is at most*

$$\begin{cases} \mathcal{O}((g(W,P,K,d) \cdot f(PV) + V)(\frac{n}{PV})^{\log_b a}), \\ \quad \text{if } \exists c > 1 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : a \cdot f(\frac{n}{b}) \geq c \cdot f(n). \\ \mathcal{O}(g(W,P,K,d) \cdot f(n) \cdot \log \frac{n}{PV} + (\frac{n}{PV})^{\log_b a} V), \\ \quad \text{if } \exists k \geq 0 : f(n) \in \Theta(n^{\log_b a} \log^k n). \\ \mathcal{O}(g(W,P,K,d) \cdot f(n)), \\ \quad \text{if } \exists c < 1 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : a \cdot f(\frac{n}{b}) \leq c \cdot f(n). \end{cases}$$

Proof Sketch. $g(W, P, K, d)$ can be factored out, so that the additional parameters are not being eliminated. In cases 1 and 2, cache faults can be prevented from level of recursion $\mathcal{O}(\log_b \frac{n}{PV})$ to the base cases by caching the translation paths to all particular input data in V pages of the TC. As a result, the factors differ from the corresponding factors of the master theorem. Furthermore, accessing the input data of the recursive calls of level $\mathcal{O}(\log_b \frac{n}{PV})$ and deeper incurs at most $\mathcal{O}((\frac{n}{PV})^{\log_b a} V)$ faults. □

The theorem also applies for out-of-place algorithms, if the used memory areas fulfill the *inclusion property*, i.e., if each memory area accessed by a recursive call is a contiguous subset of the corresponding memory area of the particular recursive caller. Details will appear in a full version of this note.

A sequential accesses pattern within a divide-and-conquer algorithm causes up to $\mathcal{O}(a^i d + \frac{m}{P})$ cache faults on the ith level of recursion, where m is the size of the memory area storing the data accessed. The following statement shows that the number decreases to $\mathcal{O}(d + \frac{m}{P})$, if a translation path of a "nearby" page is cached at the beginning of each recursive call.

Proposition 5 *Let $a \in \mathbb{N}$ and S be a sequential subsequence of the accesses of a divide-and-conquer algorithm. Consider the memory areas accessed by S at a fixed level of recursion. If*

- *the recursive calls' memory areas are disjoint, each contiguous, and contained in a contiguous memory area of overall size m,*
- *the translation path of the previous access of S is still present when the particular next access of S within the same recursive call occurs, and*
- *a translation path of any address between the currently accessed page and the first page of the $(a-1)$th previous memory area (ordered by their addresses) is present when the first access of S within any recursive call except for the one corresponding to the first memory area occurs,*

then S maximally incurs $(a+1)d + a\frac{K}{K-1}\frac{m}{P}$ cache faults on that level of recursion.

For all but the first memory accesses within a call of a recursive algorithm, an address of the current memory area may already be present. Hence, if we strengthen the third condition to require the presence of a translation path of an address of the current memory area, the upper bound amounts to $2d + 2\frac{K}{K-1}\frac{m}{P}$.

In summary, it can be stated that divide and conquer is an important design tool for VAT-algorithms.

4 Closest Pair Algorithms

In this section, we investigate the VAT-efficiency for algorithms representing different design paradigms.

4.1 Divide and Conquer

The divide-and-conquer algorithm by Bentley and Shamos [4] divides the point set by the median x-coordinate, recurses on both subsets, merges the subsets by y-coordinates, and, determines the closest pair within a certain vertical stripe around the median x-coordinate. To efficiently compute the median, implementations usually presort the input points. We can implement the out-of-place merging by five sequential scans (three interleaved scans to merge the points in an additional memory area and two interleaved scans to copy them back to the input area). In parallel to copying the points back, the points of the vertical stripe can be extracted to the beginning of the additional memory by interleaved scanning. Finally, the closest pair of points within the stripe can be found with the cost of a single scan, as each point from the stripe has do be compared to maximally seven subsequent points.

By Proposition 5, each of the seven scans incurs $\mathcal{O}(d + \frac{n}{P})$ cache faults on every level of recursion. As both memory areas used obey the inclusion property when being allocated appropriately, the calls at level of recursion $\mathcal{O}(\log \frac{n}{PW})$ and deeper maximally cause $\mathcal{O}((\frac{n}{PW})^{\log_2 2} W) = O(\frac{n}{P})$ faults (Thm. 4(2), $V \in \Theta(W)$). Therefore, the algorithm incurs $\mathcal{O}(\frac{n}{P} \log \frac{n}{PW})$ cache faults, provided that $W \geq 4d + \Omega(d)$. By (A1), their costs amount to $\mathcal{O}(\frac{n}{d} \log \frac{n}{PW}) \subseteq \mathcal{O}(n \log K)$. Consequently, the overall VAT-complexity $\mathcal{O}(n \log n)$ is dominated by the RAM-complexity. The number of cache faults can be decreased further to the translation cost of sorting $\mathcal{O}(\frac{n}{P}\lceil \frac{\log n/(PW)}{\log W/d} \rceil)$ by increasing the branching factor to $\Theta(W/d)$, similar to distribution sweeping [8].

4.2 Plane-Sweep

The plane-sweep algorithm by Hinrichs *et al.* [9] incrementally computes the closest pair by iteratively processing the input points in x-order. It maintains points already considered but still relevant ordered by their y-coordinates in a search tree. When advancing to the next point p, points not relevant anymore are deleted from the search tree, p is inserted, and is then compared to a constant number of adjacent points in the tree.

In the VAT-model, the cost of the initial sorting step is $\mathcal{O}(n \log n)$ [10]. Accessing the respectively current point and the points potentially not relevant anymore within the sorted array overall incurs $\mathcal{O}(d + \frac{n}{P})$ cache faults. Assuming a VAT-optimal search data structure, the insert, delete, and search operations cause $\mathcal{O}(n \log_P \frac{n}{W} + n \log_K \frac{n}{W} \log\log_P \frac{n}{W})$ cache faults. The overall VAT-complexity $\Theta(n \log n + \tau n(\log_P \frac{n}{W} + \log_K \frac{n}{W} \log\log_P \frac{n}{W}))$ is dominated by the translation cost of searching, provided that $\tau \in \Omega(\log K / \log\log_P \frac{n}{W})$.

4.3 Randomized Incremental Construction

The algorithm by Golin *et al.* [7] uses randomized incremental construction. Initially, the points are randomly permuted. Even a naive implementation with RAM-complexity $\Theta(n)$ incurs $\mathcal{O}(n \log_K \frac{n}{PW})$ cache faults, whereas sorting increases the RAM-complexity. Next, the algorithm iteratively inserts the points into a grid of mesh size equal to the distance of the closest pair by then. In this way, a constant number of queries to the grid and, if necessary, a rebuild suffice to process each point. As the probability of a rebuild in the ith iteration amounts to $\mathcal{O}(1/i)$, the expected overall RAM-complexity is $\Theta(n)$, if dynamic perfect hashing is used. In that case, by Proposition 2, the expected number of cache faults amounts to $\Theta(n \log_K \frac{n}{PW})$. Therefore, the translation cost of random permuting and hashing dominate the VAT-complexity.

5 Experimental Evaluation

To evaluate the practical impact of translation cost, we implemented the above closest pair algorithms in C++. The experiments were run on a single core of an Intel Core i5-4210 CPU clocked at 2.7 GHz with 3 MiB cache and 8 GiB main memory running a 64-bit Ubuntu 14.04.2 OS. The code was compiled with g++ 4.8.2 and optimization level -O3. The runtimes were averaged over 100 measurements.

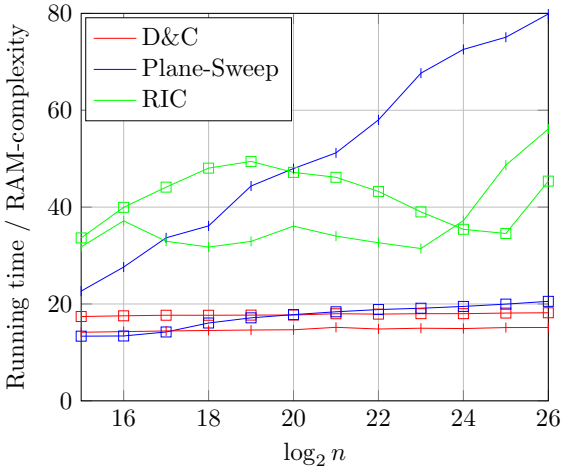


Figure 1: Normalized operation time in logarithmic scale of closest pair algorithms

The above figure illustrates the *normalized operation time*, i.e., the measured running time divided by the RAM-complexity [10]. The algorithms were applied to random point sets distributed in two different ways. The points are uniformly distributed in the unit square ($\square$) and in $\{0\} \times [0, 1)$ (depicted as $|$) respectively. In both cases, the normalized operation time of the algorithm by Bentley and Shamos is constant. Therefore, the practical runtime behavior matches the RAM-complexity, as predicted by the VAT-model. The implementation of the algorithm by Hinrichs *et al.* uses `std::set`, i.e., a balanced search tree without an optimal memory layout and with overall VAT-complexity $\mathcal{O}(\tau n \log \frac{n}{W} \log_K \frac{n}{PW})$. In case of the first distribution, it is not clear if the normalized operation time is bounded. In the second case, the normalized operation time seems to grow approximately linear in the logarithm of the input size. Thus, the worst-case runtime behavior $\mathcal{O}(n \log^2 n)$ seems to exceed the RAM-complexity and match the VAT-complexity.

We evaluated implementations of the algorithm by Golin *et al.* using hash maps of different libraries and several different hash functions. In all cases, the shape of the graph is just as irregular as the depicted (downscaled) normalized operation time of the implementation using `std::unordered_multimap`. Because of that, the runtime behavior does not entirely conform with the VAT-complexity. It seems, however, to exceed the RAM-complexity. We leave the explanation of this phenomenon as an open problem.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, Sept. 1988.

[2] R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1(3):173–189, 1972.

[3] M. A. Bender, R. Cole, and R. Raman. Exponential structures for efficient cache-oblivious algorithms. In *Proc. 29th ICALP*, pages 195–207, 2002.

[4] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proc. 8th ACM STOC*, pages 220–230, 1976.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2009.

[6] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th FOCS*, pages 285–298, 1999.

[7] M. Golin, R. Raman, C. Schwarz, and M. Smid. Simple randomized algorithms for closest pair problems. *Nordic Journal of Computing*, 2(1):3–27, Mar. 1995.

[8] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proc. 34th FOCS*, pages 714–723, 1993.

[9] K. Hinrichs, J. Nievergelt, and P. Schorn. Plane-sweep solves the closest pair problem elegantly. *Information Processing Letters*, 26(5):255–261, Jan. 1988.

[10] T. Jurkiewicz and K. Mehlhorn. The cost of address translation. In *Proc. ALENEX*, pages 148–162, 2013.

[11] T. Jurkiewicz, K. Mehlhorn, and P. K. Nicholson. Cache-oblivious VAT-algorithms. *Computing Research Repository*, abs/1404.3577, 2014.

[12] H. Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, June 1999.

Computing the maximum overlap of a disk and a piecewise circular domain under translation

Narcís Coll* Marta Fort* J. Antoni Sellarès*

Abstract

We present a GPU parallel algorithm for approximately computing the maximum overlap of a disk and a piecewise circular domain under translation. We also provide initial experimental results obtained with the implementation of our algorithm.

1 Introduction

The continuous maximal coverage problem consists in siting facilities in the continuous space to maximize coverage of regional demand. We study the one-facility case, with the assumption of uniformly distributed demand and a disk-like service area for the facility (see Figure 1 for a motivational example).

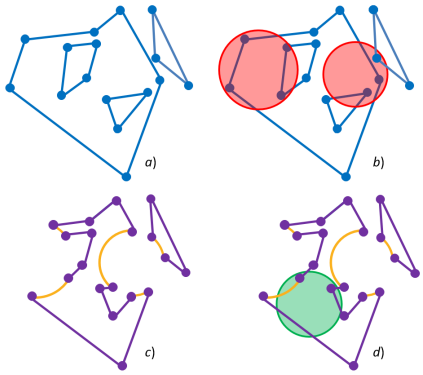


Figure 1: a) Polygonal domain to be partially covered by circular sensors; b) Domain partially covered by two circular sensors; c) Piecewise circular domain not yet covered; d) New sensor partially covering the piecewise circular domain.

A piecewise circular curve is a finite ordered list of connected circular arcs and line segments (considered as circular arcs with infinite radius). The arcs and line segments are the edges, and the points where these edges intersect are the vertices of the piecewise circular curve. A piecewise circular curve is closed if its first and last vertices coincide, and weakly simple if some pair of non-adjacent edges may intersect but the edges do not cross. A piecewise circular region is a set whose boundary is a closed weakly simple piecewise circular curve. A piecewise circular region with holes is a piecewise circular region from which the union of the interiors of a finite number of enclosed piecewise circular regions, which define the holes, has been removed. The boundaries of the enclosing piecewise circular region and the holes are pairwise disjoint, and the holes are empty. A piecewise circular domain is the union of a finite collection of non overlapping piecewise circular regions with holes.

Next, we formally define the problem to be solved. Let P be a given piecewise circular domain. For any point $q \in \mathbb{R}^2$, denote $D_r(q)$ the disk of center q and radius r. The goal is to find a location $q_0 \in \mathbb{R}^2$ which maximizes the area $A(q)$ of the overlap of $D_r(q)$ with P.

There has been some related work on this problem. Given two simple polygons P and Q with n and m vertices, respectively, Mount et al. [4] gave an algorithm to compute their maximum overlap under translation in $O(n^2m^2)$ time. Cheong et al. [1] proposed an algorithm to approximate the maximum overlap using random sampling techniques. With high probability the additive error is $\varepsilon \cdot \min\{\text{area}(P), \text{area}(Q)\}$ and the running time is $O(n + (m^2\varepsilon^{-4}\log_2 m))$. More recently, Cheng and Lam [2] presented an algorithm to approximate the maximum overlap of two polygons P and Q, built upon the framework of Cheong et al. [1]. Polygons P and Q may have multiple holes. If n denotes the total number of vertices in P and Q, the running time of the algorithm is $O(n^2\varepsilon^{-3}\log^{1.5} n \log(n/\varepsilon))$. If one of the two polygons is convex, the additive error with high probability is $\varepsilon \cdot \text{area}(P)$ and the running time can be improved to $O(n\log n + \varepsilon^{-3}\log^{2.5} n \log((\log n)/\varepsilon))$.

The prohibitive running times of the existing approximation algorithms, mainly for small ε values, motivated us to design a GPU parallel approach to efficiently find a set of approximate solutions. An initial version of this paper dealing with polygonal domains was presented in [3].

2 Overlap area computation

To solve the problem we need an efficient way to compute the area of the overlap between a disk and a piecewise circular domain. It can be computed as the area of the overlap between the disk and the outer components of the domain minus the area of overlap between the disk and each one of the holes. Along this section we provide a way to exactly and efficiently compute the area $A(q) = \text{area}(D_r(q) \bigcap R)$ of the over-

*Departament d'Informàtica, Matemàtica Aplicada i Estadística. Universitat de Girona, Spain, {coll,mfort,sellares}@ima.udg.edu.

lap of $D_r(q)$ with a piecewise circular region R without holes. This area $A(q)$ can be computed in time proportional to the number n of the vertices of R as follows. The area $A(q)$ is equal to the area of overlap between $D_r(O)$ and R', where O represents the origin and R' is the region R translated by the vector $-\overrightarrow{Oq}$. Taking into account that a circular arc or a segment intersects a circle in at most two points, the boundary of R' can be expressed as a closed piecewise curve $B = \bigcup_{i=0}^{m-1} B_i$ ($m \leq 3n$). Each curve B_i connects the points p'_i and p'_{i+1} which are vertices of R' or intersection points between $\partial D_r(O)$ and the boundary of R'. Moreover, each B_i satisfies one of the next cases:

Case 1: B_i is a piecewise circular curve exterior to $D_r(O)$ and its endpoints p'_i and p'_{i+1} are intersection points.

Case 2: B_i is a segment contained in $D_r(O)$.

Case 3: B_i is a circular arc contained in $D_r(O)$.

Consider now the curve $\overline{B} = \bigcup_{i=0}^{m-1} \overline{B}_i$ where $\overline{B}_i$ is the radial projection of B_i onto $D_r(O)$ when B_i is exterior to $D_r(O)$ and B_i otherwise. Observe that the polycurve $\overline{B}$: 1) is weakly simple; 2) can be continuously approximated by simple closed curves (see Figure 2); 3) encloses a region whose area equals $A(q)$.

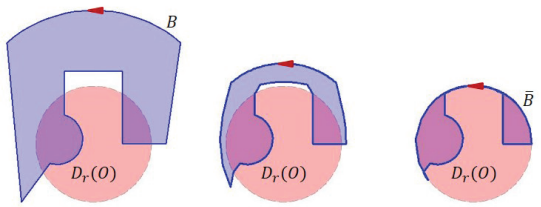


Figure 2: Approximation of $\overline{B}$ from B.

Thus, the area $A(q)$ can be computed by using Greens's theorem as follows:

$$A(q) = \frac{1}{2}\int_{\overline{B}} -ydx + xdy = \frac{1}{2}\sum_{i=0}^{m-1} I_i\,,$$

where $I_i = \int_{\overline{B}_i} -ydx + xdy$. Next, we explain how to compute the value of I_i according to the case where B_i belongs to:

Case 1: B_i is a piecewise circular curve exterior to $D_r(O)$. Let A_i be the shortest oriented arc on $D_r(O)$ that connects the point p'_{i+1} with the point p'_i and let $B_{i,i}$ be the closed simple curve determined by $B_i \cup A_i$. Then, I_i can be computed by:

$$I_i = I_{i,i} - \int_{A_i} -ydx + xdy = I_{i,i} - r^2\alpha_i\,,$$

where
$$I_{i,i} = \int_{\overline{B}_{i,i}} -ydx + xdy\,,$$

and α_i denotes the oriented angle between the vectors $\overrightarrow{Op'_{i+1}}$ and $\overrightarrow{Op'_i}$.

The result of the integral $I_{i,i}$ depends on the orientation of $B_{i,i}$ and on whether the disk $D_r(O)$ is interior or exterior to $B_{i,i}$. Let n_i be the number of intersections between B_i and the half-line with origin O in the direction of the vector $-\overrightarrow{Op'_i}$. According to n_i and α_i, there are three cases to consider (Figure 3):

1. n_i is odd and $\alpha_i > 0$. Then, $D_r(O)$ is interior to $B_{i,i}$ and $B_{i,i}$ is oriented counterclockwise. Consequently, $I_{i,i} = -2\pi r^2$ and $I_i = r^2(2\pi - \alpha_i)$.
2. n_i is odd and $\alpha_i < 0$. Then, $D_r(O)$ is interior to $B_{i,i}$ and $B_{i,i}$ is oriented clockwise. Consequently, $I_{i,i} = -2\pi r^2$ and $I_i = -r^2(2\pi + \alpha_i)$.
3. n_i is even. Then, $D_r(O)$ is exterior to $B_{i,i}$. Consequently, $I_{i,i} = 0$ and $I_i = -r^2\alpha_i$.

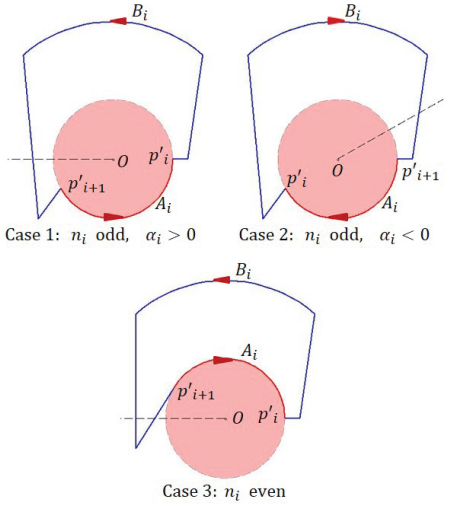


Figure 3: Exterior chain cases.

Case 2: B_i is a segment contained in $D_r(O)$. The segment $\overline{B}_i = B_i$ can be parameterized by:

$$p'_i + t(p'_{i+1} - p'_i), \quad t \in [0,1]\,.$$

Then, it holds:

$$I_i = \det(p'_i, p'_{i+1})\,.$$

Case 3: B_i is a circular arc contained in $D_r(O)$. The arc $\overline{B}_i = B_i$ can be parameterized by:

$$c_i + \cos(t)\overrightarrow{c_ip'_i} + \sin(t)\overrightarrow{c_ip'^{\perp}_i}, \quad t \in [0,\beta_i]\,,$$

where c_i is the center of the arc B_i and β_i is the oriented angle between the vectors $\overrightarrow{c_ip'_i}$ and $\overrightarrow{c_ip'_{i+1}}$. Then, it holds:

$$I_i = \det(c_i, p'_{i+1} - p'_i) + r^2\beta_i\,.$$

3 Computing the maximum overlap

In this section we describe our strategy to approximately obtain the maximum overlap of a disk and a

piecewise circular domain under translation with an ϵ-absolute error. In subsection 3.1, we present a basic approach that samples the minimum bounding box of the piece circular domain by using a uniform grid. We improve the approach locally refining the grid in subsection 3.2.

3.1 Uniform grid solution

Lemma 1 *At least a location optimizing $A(q)$ belongs to the region delimited by the convex hull $CH(P)$ of the piecewise circular domain P and, consequently, it also belongs to the minimum bounding box of P.*

The proof of Lemma 1 can be found in [3].

Lemma 1 allows us to reduce the search space for finding the maximum area of overlap. First, we sample the minimum bounding box of the piecewise circular domain P by using a uniform grid composed of square cells of side length d. We choose $d \leq \sqrt{2}r$, so that for each grid cell center c the disk $D_r(c)$ covers the cell of center c. Then, we compute the area $A(c)$ for each grid cell center c and pick a center c_0 with maximum area. To ensure that the absolute error between areas when choosing as optimal grid center c_0 instead of the optimal point q_0 is smaller than ε, we need to choose an appropriate side length d smaller than a threshold value d_ε. Let us determine d_ε.

The largest value of the absolute error $A(q_0)-A(c_0)$ occurs when the point q_0 coincides with a vertex of a square grid cell. It is bounded by the area of the lune $L(q_0) = D_r(q_0) \setminus D_r(c)$, where c is an arbitrary grid cell center, because the lune $L(q_0)$ is a subregion of $D_r(q_0) \bigcap P$ but the lune $L(c) = D_r(c) \setminus D_r(q_0)$ does not intersect $D_r(q_0) \bigcap P$ (see Figure 4a). Thus:

$$A(q_0) - A(c_0) \leq A(q_0) - A(c) \leq \text{area}(L(q_0)) \,. \quad (1)$$

In the particular worst-case in which q_0 is a vertex of the square grid cell of center c_0, taking $h = d(c_0, q_0) = d/\sqrt{(2)} \leq r$ and by using Taylor expansion, we have:

$$\text{area}(L(q_0)) = 2r^2 \arcsin\left(\frac{h}{2r}\right) + \frac{h}{2}\sqrt{4r^2 - h^2} \leq$$
$$\leq 2rh = \sqrt{2}dr \,.$$

Fixed r and $\varepsilon \in (0, 1]$, and according to (1), an ε absolute error can be guaranteed by choosing the side length d_ε of a square grid cell as:

$$d_\varepsilon = \min\left(\sqrt{2}r, \frac{\varepsilon}{\sqrt{2}r}\right) ,$$

because, in such a case:

$$A(q_0) - A(c_0) \leq \text{area}(L(q_0)) \leq \sqrt{2}d_\varepsilon r = \varepsilon \,.$$

Thus, if the bounding box of the polygon P has dimension $a \times b$, the number of vertices of the regular grid providing an ε absolute error is $(\lceil a/d_\varepsilon \rceil + 1)(\lceil b/d_\varepsilon \rceil + 1) \in O(ab(r/\varepsilon)^2)$.

3.2 Adaptive local grid refinement

If we use an initial grid of size d_ϵ over the entire bounding box of P we may waste a lot of grid cells in areas where they are not necessary. It may be computationally demanding in time and memory requirements. Using a global refinement method would have the same problems. Thus, we propose a local grid refinement method starting with a coarser grid of side length d with $\sqrt{2}r \geq d \geq d_\varepsilon$. The local grid refinement strategy identifies the cells, called parent cells, to be refined according to a two-way filtering criterium that allows us to quickly detect grid cells where it is not necessary to apply the refinement process because their points can not be optimal, Lemma 2, or are all optimal, Lemma 3. After detecting the parent cells, we determine a new smaller value for d, according to the desired ε or the maximum number of desired grid cells used per refinement step. We construct a new regular grid of child cells on each selected parent cell and we perform, for each child cell, the process we followed for the initial grid. This local refinement process can be repeated as many times as required until $d \leq d_\varepsilon$ and the desired accuracy is obtained.

Next, we give the mentioned Lemmas:

Lemma 2 *If $d \leq 2\sqrt{2}r$ and $A(c_0) - A(c) > \sqrt{2}dr$ for the center c of a grid cell g, then the optimal point q_0 does not belong to the cell g.*

Proof. From (1) we know that if the cell g of center c contains q_0 the following inequalities are fulfilled:

$$A(q_0) - A(c) \leq \text{area}(L(q_0)) \leq \sqrt{2}dr \,. \quad (2)$$

If $A(c_0) - A(c) > \sqrt{2}dr$, then also

$$A(q_0) - A(c) \geq A(c_0) - A(c) > \sqrt{2}dr \,, \quad (3)$$

thus q_0 cannot belong to g because in this case inequalities (2) and (3) are in contradiction. □

Lemma 3 *Assume that $A(c) = \text{area}(D_r(c) \cap P) = \text{area}(D_r(c)) = \pi r^2$, thus $D_r(c) \subseteq P$ and c is an optimal point. If for the center c' of each one of the eight cells adjacent to cell g we have $A(c') = \pi r^2$, then any point of the cell g is optimal.*

Proof. The union F of the set of all disks of radius r whose center belongs to the cell g is the r-offset of the cell g. Consequently, if $F \subseteq P$, then, $D_r(c') \subseteq P$ for any point c' belonging to the cell g, thus c' is an optimal point. Observe that the union of the eight disks of radius r centered in the center of the grid cells adjacent to cell g together with the disk $D_r(c)$ covers the r-offset F (see Figure 4b)). Thus, in order to have $F \subseteq P$ it suffices that $D_r(c') \subseteq P$, or equivalently $D_r(c') = \pi r^2$, for the center c' of each of the eight adjacent cells to g. □

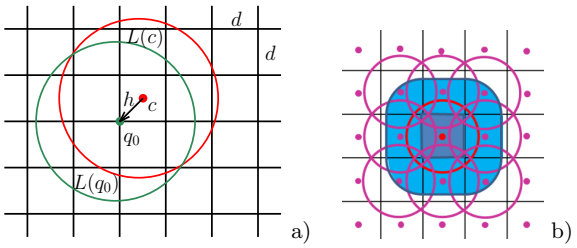


Figure 4: a) The maximum error occurs when the optimal point q_0 is a vertex of a cell. b) The union of the nine disks covers the cell offset.

4 GPU Implementation

To solve the problem in the GPU we have to transfer the piecewise circular domain to the GPU. We use: a float array storing the vertex coordinates; three integer values with the number of vertices, of enclosed components and of holes; an integer array p in which $p[i] = -1$ if the edge from vertex v_i to vertex v_{i+1} is a line segment and $p[i] = j \geq 0$ otherwise; and two extra float arrays with the radii and centers of the circular arcs, their jth element is the radius and center of the edge from v_i to v_{i+1} whenever $p[i] = j \geq 0$.

The initial overlap areas, corresponding to the circles centered at a $a \times b$ grid with $d \leq \sqrt{2}r$, are computed in parallel by finding one area per thread, the current $A(c_0)$ value is estimated using an atomic max operation. The threads in a block cooperate to store the polygon vertices information to shared memory. If a refinement step is required, we analyze the cells in parallel using the filtering criteria, parent cells are marked with a 1 and terminal cells with a 2 or a 0 depending on whether they are or cannot be optimal. It is done by using squared $B \times B$ blocks, whose threads cooperate to transfer the $(B+2) \times (B+2)$ potentially required areas to shared memory. After being marked, the N parent cells are extracted and a $k \times k$ new grid is placed in each one. We take $k = \min(32, a, b, k_\varepsilon)$ in the first step, where the value 32 is due to GPU reasons and $k_\varepsilon = d/d_\varepsilon$. In the subsequent steps $k = \min(32, k, k_\varepsilon)$. Note that at each refinement step d is divided by k becoming d/k. We add two extra rows and columns surrounding the $k \times k$ grid, that will never be refined, to be able to properly use the stop-refining criterium of Lemma 3 at this refinement step. Thus, we compute $(k+2) \times (k+2)$ areas per parent cell. Finally, the $N(k+2)^2$ areas are computed in parallel and $k \times k$ blocks are used in the next filtering step, if it is required.

5 Experimental results

We have implemented our algorithms in C++ and Cuda C and run the experiments using a Inter(R) Core(TM) i7-4790CPU with a Tesla k40 active GPU.

We have used the piece-wise circular polygon with holes of 3569 edges and considered the orange circle of $r = 3.6$ (km) that are shown in Figure 5.

We can see, represented by colored points, the cell centers analyzed during the process. The green ones are the centers of the cells having maximum area, in this case $\pi r^2 \approx 41$ (km^2). The blue points are the other analyzed centers, they are painted in a blue color gradation according to the overlap area of the circle centered on them. The darker the point the smaller the overlap area. The gradual refinement is clearly seen in the figure as well as some rounding errors in the points surrounding the regions with maximal area.

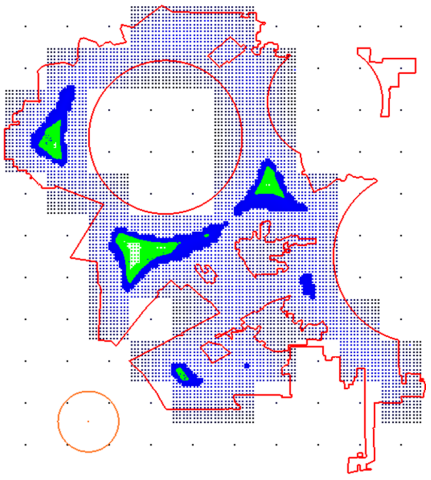


Figure 5: Piecewise circular domain with the analyzed grid cells centers, the optimal locations are marked in green. Initial grid size=11×10.

Considering $\varepsilon = 0.144$ (km^2) and a 11×10 initial grid, which corresponds to an initial grid cell size of $d = 4.9$ (km), three refinement steps are needed until a grid with cell size of 17 (m) is obtained, the algorithm takes 0.14 (s). By using a finer 352×285 initial grid, which has $d = 211$ (m), one refinement step and 0.19 (s) are needed, now the smallest d is of 26 (m).

Acknowledgments

Work partially funded by the TIN2014-52211-C2-2-R project from MEC, Spain. We also acknowledge NVIDIA Corporation for the donation of the Tesla K40 GPU.

References

[1] O. Cheong, A. Efrat, S. Har-Peled, Finding a guard that sees most and a shop that sells most, *Discrete & Computational Geometry* **37(4)** (2007) 545-563.

[2] S.W. Cheng, C.K. Lam, Shape matching under rigid motion, *Comput. Geom. Theory Appl.* **46(6)** (2013) 591–603.

[3] N. Coll, M. Fort, J.A. Sellarès, Computing the maximum overlap of a disk and a polygon with holes under translation, *XVI Encuentros de Geometría Computacional* (2015) 57–60.

[4] D.M. Mount, R. Silverman, A.Y. Wu, On the area of overlap of translated polygons, *Computer Vision and Image Understanding* **64(1)** (1996) 53–61.

Beaconless geocast protocols are interesting, even in 1D

Joachim Gudmundsson* Irina Kostitsyna† Maarten Löffler‡ Vera Sacristán§ Rodrigo I. Silveira§

Abstract

Beaconless geocast protocols are routing protocols used to send messages in mobile ad-hoc wireless networks, in which the only information available to each node is its own location. Packets get routed in a distributed manner: each node uses local decision rules based on the packet source and destination, and its own location. In this paper we analyze some of the most relevant existing protocols in a formal and structured way, focusing on two relevant 1D scenarios.

1 Introduction

In mobile ad-hoc wireless networks there is no fixed infrastructure or global knowledge about the network topology. Nodes communicate on a peer-to-peer basis, using only local information. Thus messages between nodes that are not within range of each other must be sent through other nodes acting as relay stations. An important particular case of ad-hoc wireless networks are wireless sensor networks, in which a (usually large) number of autonomous sensor nodes collaborate to collectively gather information about a certain area.

Nodes are typically mobile devices whose location and availability may change frequently, resulting in a highly dynamic environment in which routing must be done on-the-fly. Typically, messages are not sent to a particular network address, but to some or all nodes within a geographic region. This is called *geocasting*. The main pieces of information used to send a message are the location of the source node, and that of the destination region (also referred as *geocast region*), which is usually included in the actual message.

Many geocast protocols have been proposed. In general, existing protocols can be divided into two groups: those that assume that each node also knows the location of its 1-hop neighbors (i.e., all nodes within range) and those that don't. In practice, the locations of neighbors can be obtained by regularly exchanging *beacon* messages in the neighborhood. Beacons imply a significant message overhead, which prevents these methods from scaling even to medium-size networks [2]. For this reason, in this paper we are interested in the second group, the so-called *beaconless* geocast protocols.

Probably the most straightforward beaconless geocast protocol is simple flooding: each message is broadcasted to all neighbors, who in turn broadcast it to all their neighbors, and so on. Even though it is effective, the resulting message overhead is clearly unaffordable. From there on, there have been many improvements proposed. The ultimate goal is to reduce the message overhead as much as possible while still guaranteeing delivery. Due to space limitations, a proper review of all existing beaconless geocast protocols is not possible here. We refer the reader to [4] for a survey. Given the importance of geocast protocols and the many options available, several comparative studies have been presented (e.g., see those in [1, 3]) to asses the efficiency and efficacy of different methods under different scenarios. However, previous comparisons are mostly based on computer simulations.

In this paper we are interested in analyzing the behavior of beaconless geocast protocols from a theoretical and geometric perspective, since the geocast problem is inherently geometric. To that end, we present a structured overview of the main existing protocols in the literature, and identify important quality criteria to analyze them mathematically. The behavior of a geocast protocol, in general, must be analyzed in the context of a particular geometric scenario (i.e., a certain configuration of nodes). In this paper, we focus on two very fundamental geometric scenarios in 1D. Even though it is clear that the full complexity of these protocols can only be appreciated in two dimensions, we show that the 1D scenarios considered, despite their apparent simplicity, already pose interesting challenges, and already expose many of the essential differences between the protocols studied. For each scenario we analyze worst- and expected-case performance of six different protocols. The results obtained corroborate many of the findings previously obtained by simulations, and gives some insight into the difficulties of the 2-dimensional case.

2 Studied protocols

We have analyzed a selection of beaconless geocasting heuristics that are representative of different widely used strategies. These heuristics are frequently com-

*School of Information Technologies, University of Sydney, joachim.gudmundsson@gmail.com

†Dept. of Mathematics and Computer Science, Eindhoven University of Technology, i.kostitsyna@tue.nl

‡Dept. of Information and Computing Science, Universiteit Utrecht, m.loffler@uu.nl

§Dept. de Matemàtiques, Universitat Politècnica de Catalunya, {vera.sacristan,rodrigo.silveira}@upc.edu

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

bined in geocasting protocols, where the nodes follow local decision rules that are disjunctions of several different heuristic predicates. In addition, the resulting predicate is often combined via conjunction with a location predicate to control the region where each packet must travel. In this section we sketch the main characteristics of each geocast heuristics.

Simple flooding. In this protocol, when a node receives a packet, it broadcasts it—after checking that it has not broadcasted it before—and stores its ID in order to make sure it will not broadcast it again. This strategy is simple and robust, but it is non-scalable, as it produces an excessive and unnecessary network load. In the following, we describe several heuristics intended to reduce such flood load. Nevertheless, it is interesting to consider simple flooding not only for comparison purposes, but also because it is used as a building block in other protocols [3].

2.1 Restricted flooding

The following are two simple heuristics that can be considered restricted versions of flooding.

M heuristic. The MinTrans (M) heuristic explicitly controls redundancy through a parameter M: A node broadcasts a received packet if and only if the number of transmissions received for that ID is less than M. The redundant propagation allowed by the parameter M helps against problems such as message collisions and getting out from local optima.

T heuristic. The Threshold (T) heuristic uses location information for spreading the geocast propagation outward: A node retransmits a received packet if and only if the closest among all transmitters of packets with the same ID is at least a distance T from it.

2.2 Distance-based heuristics.

The previous heuristics are likely to have delivery failures in the presence of obstacles. The following protocols were designed to help solving this problem.

CD heuristic. The Center-Distance (CD) heuristic [1] relies on proximity: A node retransmits a received packet if and only if its distance to the center of the geocast region is less than that of all originators of transmissions received for the packet ID.

CD-P heuristic. This protocol [1] uses priority queues in order to further reduce the scalability problems of the CD heuristic. Each time the node can transmit, it transmits any packet that has not been transmitted at all yet (if any) or it (re)transmits, among all heard packets, the one whose transmission would give the largest reduction in distance to the center of the geocast region.

2.3 Delay-based heuristics

Some strategies to further reduce redundancy combine distances with retransmission delay.

BLR heuristic. In the Beacon-Less Routing (BLR) heuristic, each node determines when to retransmit a received packet based on a dynamic forwarding delay function valued in $[0, \mathrm{MD}]$, for MD a constant representing the maximum delay. The node retransmits the package after such delay, unless some other node does it before, in which case the retransmission is canceled. Three delay functions have been suggested in [2], based on the following parameters: r (transmission range), p (progress towards destination of the orthogonal projection of the current node onto the previous_node-destination line), and d (distance from current node to the source-destination line). The proposed variants are: $\mathrm{delay}_1 = \mathrm{MD}\ \frac{r-p}{r}$, $\mathrm{delay}_2 = \mathrm{MD}\ \frac{p}{r}$, and $\mathrm{delay}_3 = \mathrm{MD}\ \frac{e^{\sqrt{p^2+d^2}}}{e}$.

GeRaF heuristic. Based on distance, the Geometric Random Forwarding (GeRaF) protocol [6] logically divides the area around the destination of a packet p into n_p areas $\mathcal{A}_1, \ldots, \mathcal{A}_{n_p}$, where in $\mathcal{A}_1$ are all nodes closest to the destination, and so on. Once p is transmitted, up to n_p phases start, during which all nodes listen during a fixed amount of time. In the first phase, nodes in region $\mathcal{A}_1$ get to reply. If only one node replies, then that one will forward the message. If there are more, some collision resolution scheme must be used. If there is no reply, then it is the turn to reply for nodes in region $\mathcal{A}_2$. This process continues until some node in the non-empty region closest to destination replies.

Greedy routing (beaconless version). Greedy routing does not always guarantee delivery. Nevertheless, a greedy routing strategy is often used as building block of geocast protocols. For this reason we also consider greedy routing in our analysis. One example is Geographic Distance Routing (GeDiR) [5]. GeDir requires to know the position of all neighbors of a node: it is a greedy algorithm that always forwards the message to the neighbor of the current node whose distance to the destination is minimum.

This strategy can be made beaconless by a delay function based on the following parameters: r (transmission range), d (distance from previous node to destinations), and x (distance from current node to destination), by using delay function $\mathrm{delay}_4 = \mathrm{MD}\ \frac{x+r-d}{2r}$.

This strategy tries to get out of local minima by sending the packet to the best positioned neighbor, even if it is not closer to destination than the sender.

All protocols described in this section include a rule that states what to do if a node receives a message already in its queue. One option, like in BLR, is to always cancel the transmission of a message received

Figure 1: Illustration of the scenarios. Left: with unbounded reach, the k messages arrive immediately to all nodes, but that does not prevent intermediate nodes from forwarding the messages. Right: with range $r = 2$, the messages sent from node 0 only reach up to node 2, so forwards are necessary to reach the target, $n + 1$.

twice. Another option also used in practice is to cancel only if the sender of the duplicate message is closer to the destination than the current node.

3 The 1D analysis

In this section we study two fundamental scenarios in 1D, in which the leftmost of $n+2$ nodes sends k packets to the rightmost node (i.e., to a goal region which only contains the rightmost node). Each packet contains the position of its last (re)sender and its destination. Each node stores all received packets in a queue which is managed in one way or another depending on the protocol used. For simplicity, nodes are evenly spread at unit distance along the line. The n intermediate nodes form a dense bottleneck, a situation that can easily arise in practice. Once the transmissions start, collisions may happen. In order to cope with this problem, we assume fair medium access, i.e., the transmission is done by rounds, and in each round each node that has some packet to transmit has the same probability to transmit it.

Several aspects have to be taken into account when comparing the behavior of different protocols. The *success rate* measures the fraction of sent messages that actually reach the target. For those that arrive, the *hop count* indicates how many steps (forwards) are needed. In this paper we only focus on what we consider to be the most significant measure within this context, *RecMess*, which is defined as the maximum number of packets that a node receives. This parameter measures the work or energy consumption for a node, as well as the overall network load and therefore, its congestion.

Due to space limitation we only give the ideas of the proofs of the theorems.

3.1 Unbounded reach scenario

In the *unbounded reach scenario*, all nodes are within the range of each other. This setting recreates a rather frequent situation in which many messages must go through a high-density area.

Simple flooding. Under this protocol, all the nodes will receive and retransmit all the packets: $\mathbf{RecMess} = nk$.

M heuristic. By definition, every node receives every message at most M times: $\mathbf{RecMess} = Mk$.

T heuristic. Due to the unit-distance distribution of the nodes and assuming $T \in \mathbb{N}$, when a node transmits a packet, then this packet is deleted from the queues of its T left and its T right neighbors. Thus: $\lceil \frac{n}{2T} \rceil k \leq \mathbf{RecMess} \leq \lceil \frac{n}{T} \rceil k$.

CD and CD-P heuristic. We note that **RecMess** in CD heuristic is higher than in CD-P heuristic. And move to analyzing the CD-P heuristic. We prove the following theorem.

Theorem 1 *In the unbounded reach scenario under the CD-P heuristic,* $\boldsymbol{RecMess} = O(k \log n)$.

Proof idea. To prove this theorem we represent the message queues of the nodes as columns in a $k \times n$ table, and introduce k random variables that represent the rows' lengths. We bound the size of the longest row after every message transmission using probabilistic analysis of the kth order statistic.

Delay-based. We assume that the delay is chosen such that it increases by exactly one time step per node; that is, MD $= r$. The nodes delete messages from their queues when they receive them for the second time. Thus, every message is retransmitted only once, no matter which delay function is used. Therefore, $\mathbf{RecMess} = k$.

In the variant in which the nodes delete messages from their queues only when they receive a duplicate from a node that is closer to the destination, we get:

$$\mathbf{RecMess} = \begin{cases} 2^k & \text{if } k < \log n\,, \\ n + n(k - \log n) & \text{if } k \geq \log n\,. \end{cases}$$

3.2 Bounded reach scenario

In the *bounded reach scenario* each node can communicate with r neighbors to its left and r to its right, for some parameter r. This scenario generalizes the previous one, allowing to evaluate the effect that node density (indirectly related to r) has on the different protocols.

Lower bound. Any heuristic will need at least $\frac{kn}{r}$ retransmissions for all messages to reach the destination, as a message cannot progress by more than r nodes at a time. Every node receives a fraction $\Theta(\frac{r}{n})$ of all the messages, therefore, $\mathbf{RecMess} = \Omega(k)$.

Simple flooding. Under this protocol, every node will receive each message from at most $2r$ of its neighbors: **RecMess** $= O(rk)$.

M heuristic. If $2r \leq M$ this protocol is equivalent to the previous one. If $2r > M$ it is equivalent to the M heuristic for the unbounded reach scenario. Therefore **RecMess** $= O(\min(M, 2r)k)$.

T heuristic. If $T \geq r$, no packet will ever be forwarded. If $T < r$, then each node u can receive a packet from at most $2r$ nodes. Each time it receives one, at least T and at most $2T$ of the nodes within reach of u delete the packet from their queues. Thus: **RecMess** $= O(\frac{rk}{T})$.

CD and CD-P heuristics. Consider a $n \times k$ table, where columns represent message queues of the nodes, and rows represent the messages that are in the queues. All messages start in the leftmost r columns. Whenever a node retransmits a message, it gets deleted from all nodes to its left and added to the $r-1$ nodes to its right. Thus, each message is always present in exactly r consecutive nodes (except at the end of the process).

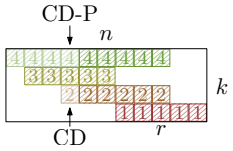


Figure 2: CD vs CD-P in bounded reach scenario.

When a node gets to retransmit, it picks the message with the lowest ID from its queue in CD heuristic, and with the highest ID in CD-P heuristic. Figure 2 illustrates the difference. On average, the progress a message makes to the destination when it gets retransmitted under the CD heuristic is smaller than under the CD-P. Thus, **RecMess** of CD heuristic is again greater than that of CD-P heuristic.

Theorem 2 *In the bounded reach scenario under the CD heuristic,* ***RecMess*** $= O(k^{3/2})$.

Proof idea. To prove this theorem we show that the expected progress a message makes when it is retransmitted is greater than $\frac{r}{\sqrt{k}+1}$, and the bound on the **RecMess** follows.

In contrast, the CD-P heuristic is optimal up to a constant factor with respect to **RecMess**.

Theorem 3 *In the bounded reach scenario under the CD-P heuristic,* ***RecMess*** $= \Theta(k)$.

Proof idea. We observe that the average progress a message makes on its way to the destination at each retransmission is greater than $\frac{r}{2}$. From which we deduce that **RecMess** $= O(k)$.

Delay-based. We discuss the case where delay_1 is used. The other functions can be analyzed in a similar way. Since all messages get deleted from its queue when heard by a node for the second time, **RecMess** $= O(\frac{nk}{r})$.

4 Concluding remarks

Beaconless geocast protocols are used in practice in 2D scenarios. They differ from the 1D ones in a few but important characteristics: the destination of a packet is defined as a region where more than one node may happen to be located; obstacles (like buildings, which cannot be traversed by the transmission signal) need to be surrounded, and local optimization strategies fail to guarantee delivery. Therefore, combinations of different strategies need to be used in order to achieve delivery guarantees and, at the same time, keep the network load within reasonable bounds. The network load analysis in this cases is difficult, and almost only experimental results exist. This is why we have started studying the 1D case. It has shown to be less trivial than we expected. Indeed, all protocols give rise to different load bounds, the CD and the CD-P heuristics being particularly tricky to analyze.

Acknowledgments I.K. and M.L. were supported by the Netherlands Organisation for Scientific Research (NWO) under projects no. 639.023.208 and 639.021.123 respectively. V.S. and R.S. were partially supported by projects MINECO MTM2015-63791-R and Gen. Cat. DGR 2014SGR46. R.S. was also supported by MINECO through the Ramón y Cajal program.

References

[1] R. J. Hall. An improved geocast for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 10(2):254–266, 2011.

[2] M. Heissenbüttel, T. Braun, T. Bernoulli, and M. Wälchli. BLR: Beacon-less routing algorithm for mobile ad-hoc networks. *Computer Communications*, 27(11):1076–1086, 2004.

[3] C. Maihöfer. A survey of geocast routing protocols. *IEEE Communications Surveys and Tutorials*, 6(1-4):32–42, 2004.

[4] S. Rührup. Theory and practice of geographic routing. In *Ad Hoc and Sensor Wireless Networks: Architectures, Algorithms and Protocols*. Bentham Science, 2009.

[5] I. Stojmenovic and X. Lin. Loop-free hybrid single-path/flooding routing algorithms with guaranteed delivery for wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1023–1032, 2001.

[6] M. Zorzi. A new contention-based mac protocol for geographic forwarding in ad hoc and sensor networks. In *IEEE Int. Conference on Communications (ICC)*, pages 3481–3485, 2004.

Computing Minimum-Link Separating Polygons in Practice*

Moritz Baum† Thomas Bläsius‡ Andreas Gemsa† Ignaz Rutter† Franziska Wegner†

Abstract

We tackle the problem of separating two given sets of polygons by a polygon with minimum number of segments. As the complexity in our specific setting is unknown, we propose heuristics that are simple enough to be implemented in practice. A key ingredient is a new practical linear-time algorithm for minimum-link paths in simple polygons. Experiments in a challenging realistic setting show excellent performance of our algorithms in practice.

1 Introduction

We study a problem motivated by isocontour visualization in road networks, where the goal is to separate the *reachable* subgraph for a given resource limit from the remaining *unreachable* part. Given an embedding of (a planarization of) such a network into the plane, the geometric subproblem is to separate reachable and unreachable *boundaries* by a simple polygon. We consider three objectives for such *range polygons*. They must be exact (i. e., correctly separate the boundaries); they should be of low *complexity* (i. e., have few segments) for an appealing visualization and efficient rendering; the algorithms should be fast in practice, even on large inputs. In this extended abstract, we focus on geometric aspects of this problem; see the full version for omitted details and proofs [1].

Fig. 1 shows an example of a *border region* B, the input of the geometric subproblem. It is defined by two sets R and U of hole-free plane polygons, containing boundaries of the reachable and unreachable part, respectively. We seek a simple polygon with minimum number of links that separates U from R. In general, this is an $\mathcal{NP}$-complete problem [3]. In our case, we have $|R| = 1$ since the reachable part is connected by definition, which, to the best of our knowledge, yields an unresolved open problem [3]. First, consider a border region B with $|R| = |U| = 1$. A polygon of minimum complexity that separates the polygons can be found in $O(n \log n)$ time [7]. However, the algorithm is rather involved and requires computation of several minimum-link paths. We propose a simpler algorithm

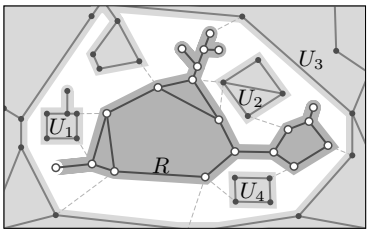


Figure 1: A border region (white area) with a reachable boundary R and an unreachable boundary U with components U_1 to U_4. Shaded areas show reachable (dark gray) and unreachable (light gray) parts.

that uses at most two additional segments, runs in linear time, and requires a single run of a minimum-link path algorithm. It adds an edge e to B that connects both boundaries. In the resulting polygon B', it computes a path with minimum number of segments that connects the two sides of e. The algorithm of Suri [6] computes such a minimum-link path π in linear time. We obtain a separating polygon S by connecting the endpoints of π along e. For practical performance, Section 2 proposes a simpler linear-time algorithm for the key ingredient of our approach, the computation of a minimum-link path. Section 3 then covers the general case of $|U| \geq 1$. Since its complexity is unknown, we focus on heuristic approaches that work well in practice, but do not give guarantees on the complexity of the resulting range polygons. We evaluate our algorithms on realistic input in Section 4.

2 A Practical Minimum-Link Path Algorithm

We address the subproblem of computing a *minimum-link path* (a polygonal path with minimum number of segments) between two edges a and b of a simple polygon P that lies in the interior of P. The algorithm of Suri [6] starts by triangulating the input polygon. In our scenario, we preprocess this step by triangulating all faces of the planar input graph only once. Afterwards, in each step of Suri's algorithm a *window* (which we define in a moment) is computed. To this end, several calls to a subroutine computing visibilty polygons are necessary. While this is sufficient to prove linear running time, it seems wasteful from a practical point of view. In the following, we present a simpler linear-time algorithm for computing the windows. It can be seen as a generalization of an algo-

*Partially supported by the EU FP7 under grant agreement no. 609026 (project MOVESMART)

†Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany. Email: first.last@kit.edu

‡Karlsruhe Institute of Technology / Hasso Plattner Institute, Germany. Email: thomas.blaesius@hpi.de

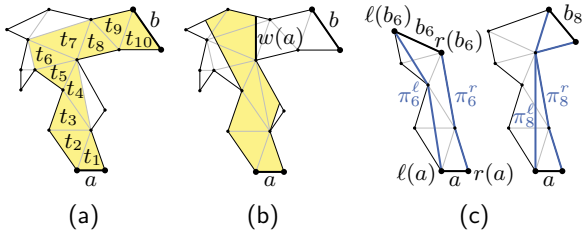


Figure 2: (a) Important triangles. (b) The window $w(a)$ is an edge of $V(a)$ (shaded). (c) The shortest paths (blue) intersect for $i = 8$ but not for $i = 6$.

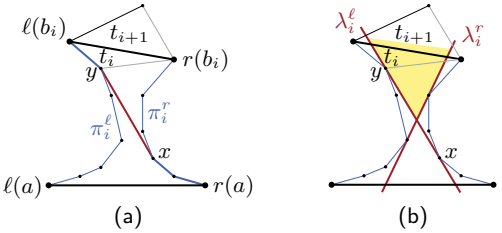


Figure 3: (a) Shortest path from $r(a)$ to $\ell(b_i)$. (b) Visibility lines spanning the (shaded) visibility cone.

rithm for approximating piecewise linear functions [4].

Let G be the (embedded) graph obtained by triangulating P. Let t_a and t_b be the triangles incident to a and b, respectively. The (weak) dual graph of G has a unique path $t_a = t_1, t_2, \ldots, t_{k-1}, t_k = t_b$ from t_a to t_b; see Fig. 2a. These triangles are *important* and their position in the path is their *index*. The *visibility polygon* $V(a)$ of the edge a in P is the polygon that contains a point p if and only if there is a point q on a such that the line segment pq lies inside P. Let j be the highest index such that the intersection of the triangle t_j with $V(a)$ is not empty. The *window* $w(a)$ is the edge of $V(a)$ that intersects t_j closest (wrt. minimum Euclidean distance) to the edge between t_j and t_{j+1}; see Fig. 2b. Note that $w(a)$ separates the polygon P into two parts. Let P' be the part containing b. A minimum-link path from a to b in P is obtained by adding an edge from a to $w(a)$ to a minimum-link path from $w(a)$ to b in P'. Thus, the next window is computed in P' starting from $w(a)$.

We describe how to compute the first window. Let G_i be the subgraph of G induced by the triangles $t_1, \ldots, t_i$ and let P_i be the polygon bounding the outer face of G_i. Then P_i has two special edges, namely a and the edge shared by t_i and t_{i+1}, called b_i. Let $\ell(a)$ and $r(a)$, and $\ell(b_i)$ and $r(b_i)$ be the endpoints of a and b_i, respectively, such that their clockwise order is $r(a)$, $\ell(a)$, $\ell(b_i)$, $r(b_i)$; see Fig. 2c. The *left shortest path* π_i^ℓ is the shortest polygonal path (wrt. Euclidean length) in P_i that connects $\ell(a)$ with $\ell(b_i)$. The *right shortest path* π_i^r is defined symmetrically; see Fig. 2c. One can show that t_{i+1} is (partially) visible from a if and only if the left and right shortest paths in P_i have empty intersection. Moreover, if these paths do not intersect, they are *outward convex*, i. e., π_i^ℓ and π_i^r have only left and right bends, respectively [2]. These two paths together with a and b_i are called *hourglass*. To keep track of parts of t_{i+1} visible from a, we use two visibility lines. To define them, consider the shortest path in the hourglass connecting $r(a)$ with $\ell(b_i)$; see Fig. 3a. It is the concatenation of a prefix of π_i^r, a line segment between vertices x and y, and a suffix of π_i^ℓ. The straight line through x and y is the *left visibility line* denoted by λ_i^ℓ. The *right visibility line* λ_i^r is defined symmetrically. The region between λ_i^ℓ and λ_i^r is the *visibility cone*; see Fig. 3b. A point in t_{i+1} is visible from a if and only if it lies in the visibility cone.

These observations justify the following approach for computing the window $w(a)$. We iteratively increase i until the left and the right shortest path of the polygon P_i intersect. We then know that the triangle t_{i+1} is no longer visible. As the left and the right shortest paths did not intersect in P_{i-1}, the triangle t_i is visible from a. Recall that $w(a)$ is the edge of the visibility polygon $V(a)$ that intersects t_i closest to the edge between t_i and t_{i+1}. Thus, $w(a)$ is a segment of one of the two visibility lines. It remains to fill out the details (how to compute the paths and the visibility lines) and describe later steps, when we start at a window instead of an edge.

We start with the details. Assume the triangle t_i is still visible from a, i.e., π_{i-1}^ℓ and π_{i-1}^r do not intersect. Assume further that we computed the left and right shortest paths π_{i-1}^ℓ and π_{i-1}^r as well as the visibility lines λ_{i-1}^ℓ and λ_{i-1}^r in a previous step. Without loss of generality, let the three corners of t_i be $\ell(b_{i-1})$, $\ell(b_i)$, and $r(b_i) = r(b_{i-1})$ (as in Fig. 4). There are three possibilities; see Fig. 4. The new vertex $\ell(b_i)$ lies either in the visibility cone spanned by λ_{i-1}^ℓ and λ_{i-1}^r, to the left of λ_{i-1}^ℓ, or to the right of λ_{i-1}^r. We know that a point in t_i is visible from a if and only if it lies inside the visibility cone. Thus, the edge b_i between t_i and t_{i+1} is no longer visible if and only if the new vertex $\ell(b_i)$ lies to the right of λ_{i-1}^r; see Fig. 4e. In this case, we can stop and output the desired window $w(a)$, which is a segment of λ_{i-1}^r; see Fig. 4f. In the other two cases (Fig. 4a and Fig. 4c), we have to compute the new left and right shortest paths π_i^ℓ and π_i^r and the visibility lines λ_i^ℓ and λ_i^r (Fig. 4b and Fig. 4d). Note that the right shortest path and the right visibility line remain unchanged, i.e., $\pi_i^r = \pi_{i-1}^r$ and $\lambda_i^r = \lambda_{i-1}^r$. The new path π_i^ℓ is obtained by concatenating the prefix of π_{i-1}^ℓ with endpoint x and the segment from x to $\ell(b_i)$, where x is the latest vertex on π_{i-1}^ℓ such that the result is outward convex. To get the new left visibility line λ_i^ℓ, we distinguish the two remaining cases. If $\ell(b_i)$ lies to the left of λ_{i-1}^ℓ (Fig. 4c), we obtain $\lambda_i^\ell = \lambda_{i-1}^\ell$; see Fig. 4d. If $\ell(b_i)$ lies in the visibility cone (Fig. 4a), the

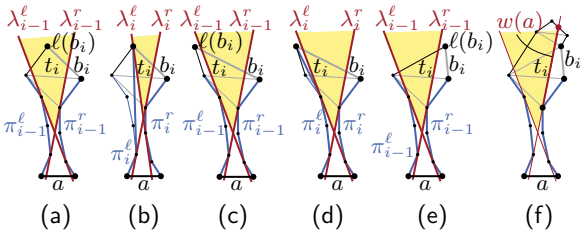


Figure 4: The three cases for the position of $\ell(b_i)$ and the updated shortest paths and visibility lines.

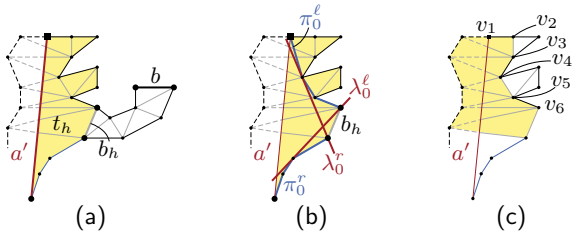


Figure 5: Initial steps for computing the next window, when starting at the previous window a'.

visibility line changes. Let x be the latest vertex on π_i^r such that the concatenation of the subpath from $r(a)$ to x with the segment from x to the new vertex $\ell(b_i)$ is outward convex. Then λ_i^ℓ is the line through x and $\ell(b_i)$; see Fig. 4b. One can show that the point x where λ_i^ℓ intersects π_i^r moves forward along π_i^r during the algorithm (which is relevant for the running time).

Lemma 1 *Let t_h be the triangle with the highest index that is visible from a. Then our algorithm computes the first window $w(a)$ in $O(h)$ time.*

The window $a' = w(a)$ separates P into two smaller polygons. Let P' be the part including the edge b. To get the next window $w(a')$, we have to apply the above procedure to P' starting with a'. However, this would require to partially retriangulate P'. More precisely, let t_h be the triangle with highest index that is visible from a; see Fig. 5a. Then b_h separates P' into an initial part P_0' and the rest (having b on its boundary). The latter part is properly triangulated, but P_0' is not. While we could retriangulate P_0', this would require an efficient subroutine for triangulation and a dynamic data structure. Instead, we propose a much simpler method for computing the next window.

The idea is to compute shortest paths in P_0' from $\ell(a')$ to $\ell(b_h)$ and from $r(a')$ to $r(b_h)$; see Fig. 5b. We denote these paths by π_0^ℓ and π_0^r, respectively. Moreover, we compute the corresponding visibility lines λ_0^ℓ and λ_0^r. Afterwards, we can continue with the correctly triangulated part as before. Concerning the shortest paths, note that the right shortest path π_0^r is a suffix of the previous right shortest path, which we already know. For the left shortest path π_0^ℓ, consider the polygon induced by the triangles intersected by a'; see Fig. 5c. Let $v_1, \ldots, v_g$ be the sequence of vertices on the outer face of this polygon (in clockwise direction) from $\ell(a') = v_1$ to $\ell(b_h) = v_g$. To obtain π_0^ℓ, we start with an empty path and iteratively append the next vertex from this sequence while maintaining the path's outward convexity by successively removing the second to last vertex if necessary. It remains to compute the initial visibility lines λ_0^ℓ and λ_0^r. Note that the whole edge b_h is visible from a', since a' intersects the triangle t_h. It follows that λ_0^ℓ is the line that goes through $\ell(b_h)$ and through the unique vertex on π_0^r such that λ_0^ℓ is tangent to π_0^r; see Fig. 5b. The same holds for the right visibility line. With these insights, it is not hard to compute the paths π_0^ℓ and π_0^r and the corresponding visibility lines in $O(|P_0'|)$ time.

We compute subsequent windows until we find the last edge b. A minimum-link path π is obtained by connecting each window $w(a)$ to its first edge a by a straight line [6]. Lemma 1 and our considerations concerning initial paths imply the following theorem.

Theorem 2 *Given two edges a and b of a simple polygon P, our algorithm computes a minimum-link path from a to b contained in P in linear time.*

3 Heuristics for the General Case

We outline heuristics for the case of $|U| \geq 1$. Fig. 6 sketches results in a small example. First, `RP-RC` (*range polygon*, extracted *reachable components*) simply returns the reachable boundary R. This approach is similar to previous algorithms for isochrones [5]. Second, `RP-TS` (*triangular separators*) uses the triangulation to separate B along edges for which both endpoints are in R. The modified instances consist of single unreachable components that are separated by the algorithm from Section 2. Third, `RP-CU` (*connecting unreachable* components) inserts new edges that connect the components of U to create an instance with $|U| = 1$. Finally, `RP-SI` (*self-intersecting* polygons) modifies the approach described in Section 2 to compute a possibly self-intersecting minimum-link path separating R and U. To obtain the range polygon, it is rearranged at intersections.

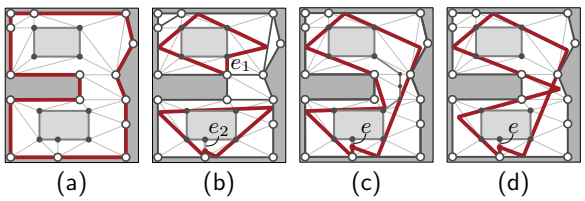


Figure 6: Results (red) of `RP-RC` (a), `RP-TS` (b), `RP-CU` (c), and `RP-SI` (d), starting at indicated edges.

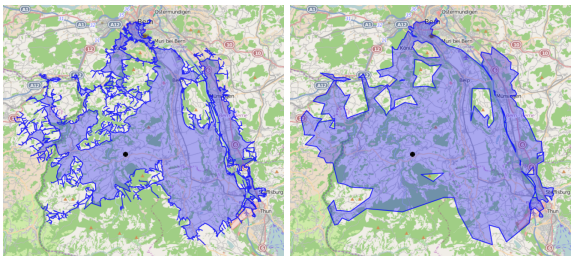


Figure 7: Real-world example of isocontours showing the range of an electric vehicles at the black disk (near Bern, Switzerland) and a state of charge of 2 kWh.

4 Evaluation

We evaluate our approaches (implemented in C++) on a graph representing the road network of Europe, with 22 million vertices and 52 million edges. Fig. 7 shows isocontours visualizing the range of an electric vehicle. The result of `RP-RC` (left) has more than 10 000 segments, even in this medium-range example. The result of `RP-CU` (right) uses much fewer (416) segments to represent the same isocontour. Note that the isocontour contains holes, due to unreachable high-ground areas. Hence, it has several border regions.

For our analysis, we focus on large ranges (roughly 500 km with 85 kWh batteries). Table 1 shows results of all heuristics from Section 3, averaged for 1 000 queries from sources picked uniformly at random. Timings include extraction of the border region from the input graph, which is part of the work required in our scenario. For `RP-SI`, we report figures for unprocessed polygons with self-intersections. Running times are well below 30 ms for all approaches. The simpler heuristics, `RP-RC` and `RP-TS`, are faster by a factor of 2 to 3. However, polygons computed by `RP-RC` have a much higher complexity (by about a factor of 50). Results of `RP-TS` have low complexity, but the triangular separation increases the number of components significantly (all other approaches have the minimum number of components, i. e., the number of border regions). For `RP-CU` and `RP-SI`, the additional effort pays off, as they keep complexity close to the optimum (off by at most 6 % according to a lower bound induced by the results of `RP-SI`).

Table 1: Results for isocontours. We show the number of components of the result (Cp.), complexity (Seg.), self-intersections (Int.), and running time in ms.

Algorithm	Cp.	Seg.	Int.	Time
`RP-RC`	131	92 554	—	9.46
`RP-TS`	219	1 973	—	7.78
`RP-CU`	131	1 820	—	25.11
`RP-SI`	131	1 781	15.06	22.25

Table 2: Minimum-link path algorithm performance. We report input complexity ($|P|$), visited triangles (v. Tr.), links in the result (Seg.), and time in ms.

Scenario	$\|P\|$	v. Tr.	Seg.	Time
EV, 16 kWh	134 049	9 334	416	0.72
Iso, 60 min	135 112	11 965	701	1.03
EV, 85 kWh	357 335	33 030	1 329	3.14
Iso, 500 min	637 224	69 398	3 204	6.57

We evaluate the minimum-link path algorithm from Section 2 in four scenarios (ranges for 16 kWh and 85 kWh batteries and isochrones for 60 and 500 minutes). For each of 1 000 random queries, we modified the largest border region such that $|U| = 1$ (using `RP-CU`). Then, we added an edge connecting the two components and computed a minimum-link path between its two sides. Recall that triangulation of the input is part of preprocessing, hence it is not reported in the table. As Table 2 shows, the running time increases with input complexity. However, our algorithm runs in less than 10 milliseconds on average in all cases. Isochrone scenarios are slightly harder due to the different shape of the border regions.

Overall, we see that our approaches are suitable for interactive applications on inputs of continental scale.

Acknowledgments. We thank Roman Prutkin for interesting discussions.

References

[1] M. Baum, T. Bläsius, A. Gemsa, I. Rutter, and F. Wegner. Scalable Isocontour Visualization in Road Networks via Minimum-Link Paths. Technical Report abs/1602.01777, ArXiv e-prints, 2016.

[2] L. J. Guibas, J. E. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[3] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating Polygons and Subdivisions with Minimum-Link Paths. *International Journal of Computational Geometry & Applications*, 3(4):383–415, 1993.

[4] H. Imai and M. Iri. An Optimal Algorithm for Approximating a Piecewise Linear Function. *Journal of Information Processing*, 9(3):159–162, 1987.

[5] S. Marciuska and J. Gamper. Determining Objects within Isochrones in Spatial Network Databases. In *Advances in Databases and Information Systems*, volume 6295 of *LNCS*, pages 392–405. Springer, 2010.

[6] S. Suri. A Linear Time Algorithm for Minimum Link Paths Inside a Simple Polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

[7] C. A. Wang. Finding Minimal Nested Polygons. *BIT Numerical Mathematics*, 31(2):230–236, 1991.

Computing Pretropisms for the Cyclic n-Roots Problem*

Jeff Sommars† Jan Verschelde‡

Abstract

The cyclic n-roots problem is an important benchmark problem for polynomial system solvers. We consider the pruning of cone intersections for a polyhedral method to compute series for the solution curves.

1 Introduction

The cyclic n-roots problem asks for the solutions of a polynomial system, commonly formulated as

$$\begin{cases} x_0 + x_1 + \cdots + x_{n-1} = 0 \\ i = 2, 4, \ldots, n-1 : \sum_{j=0}^{n-1} \prod_{k=j}^{j+i-1} x_{k \bmod n} = 0 \\ x_0 x_1 x_2 \cdots x_{n-1} - 1 = 0. \end{cases} \quad (1)$$

This problem is important in the study of biunimodular vectors, a notion that traces back to Gauss, as stated in [10]. In [3], Backelin showed that if n has a divisor that is a square, i.e. if d^2 divides n for $d \geq 2$, then there are infinitely many cyclic n-roots. The conjecture of Björck and Saffari [5], [10, Conjecture 1.1] is that if n is not divisible by a square, then the set of cyclic n-roots is finite.

As shown in [1], the result of Backelin can be recovered by polyhedral methods. Polyhedral methods to solve a polynomial system consider the Newton polytopes of the polynomials in the system. The *Newton polytope* of a polynomial in several variables is the convex hull of the exponent tuples of the monomials that appear with nonzero coefficient in the polynomial. Looking for positive dimensional solution sets, we look for series developments of the solutions, and in particular we look for Puiseux series. The leading exponents of Puiseux series are called *tropisms*. A *pretropism* is a vector that forms the minimal inner product with a face of every one of the given polytopes, where none of the faces are 0-faces. Pretropisms are candidates for being tropisms, but not every pretropism is a tropism, as pretropisms depend only on the Newton polytopes of the system, see e.g. [13] for a mathematical background on tropical algebraic geometry.

*This material is based upon work supported by the National Science Foundation under Grant No. 1440534.

†Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, sommars1@uic.edu

‡Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, janv@uic.edu.

Our problem can thus be stated as follows. Given a tuple of Newton polytopes, compute all pretropisms. In [15] we examined the case where all polytopes are in general position with respect to each other. In this paper we focus on the Newton polytopes of the cyclic n-roots problem.

Prior and related work. In [6], the computation of pretropisms is defined as the common refinement of the normal fans of the Newton polytopes [18]. The software Gfan [12] relies on cddlib [11] in its application of reverse search algorithms [2].

2 Pruning Cone Intersections

To introduce our algorithms, consider Figure 1. For three Newton polytopes (P_1, P_2, P_3), the leaves of the trees represent cones of pretropisms. Nodes without children that are not leaves correspond to cone intersections that contain only the zero dimensional cone.

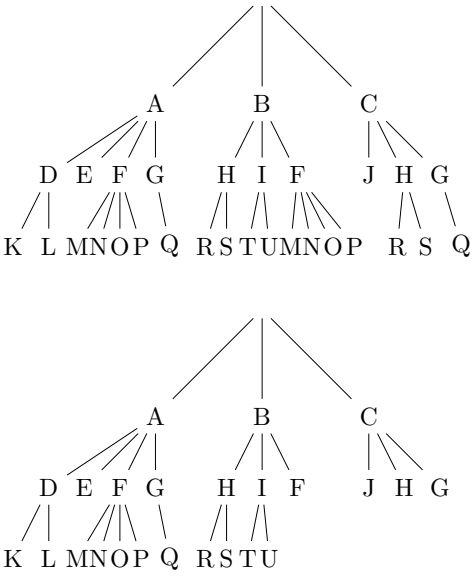


Figure 1: Nodes A, B, C represent cones to P_1. Intersections of those cones with the cones of P_2 are represented by nodes D through J. Duplicate nodes are removed from the second tree.

The removal of duplicate nodes eliminates many cone intersections at deeper levels in the tree.

3 Algorithms

Our algorithm takes as input the edge skeletons of a set of Newton polytopes; the edge skeleton of a polytope can be computed by a polynomial-time algorithm, presented in [9]. In our implementation, we use edge objects that have vertices, references to their neighboring edges, and the cone of the set of inner normals of all of the facets on which the edge rests. Note that since the cyclic-n polytopes are not all full dimensional, we included generating rays of the lineality spaces as needed.

Algorithm 2 sketches the outline of the algorithm to compute all pretropisms of a tuple of n polytopes. Along the lines of the gift wrapping algorithm, for every edge of the first polytope we take the plane that contains this edge and consider where this plane touches the second polytope. Algorithm 1 starts exploring the edge skeleton defined by the edges connected to the vertices in this touching plane.

Algorithm 1 Explores the skeleton of edges to find pretropisms of a polytope P and a cone C.

```
1: function ExploreEdgeSkeleton(P, C)
2:     r := a random ray inside C
3:     m := min{⟨a, r⟩, a ∈ P}
4:     in_r(P) := {a ∈ P, ⟨a, r⟩ = m}
5:     EdgesToTest := edges e of P: e ∩ in_r(P) ≠ ∅
6:     Cones := ∅
7:     TestedEdges := ∅
8:     while EdgesToTest ≠ ∅ do
9:         E := pop an edge from EdgesToTest
10:        C_E := normal cone to E
11:        ShouldAddCone := False
12:        if C_E contains C then
13:            ConeToAdd := C
14:            ShouldAddCone := True
15:        else if C ∩ C_E ≠ {0} then
16:            ConeToAdd := C ∩ C_E
17:            ShouldAddCone := True
18:        end if
19:        if ShouldAddCone then
20:            Cones := Cones ∪ ConeToAdd
21:            Edges := Edges ∪ E
22:            for each neighboring edge e of E do
23:                if e ∉ TestedEdges then
24:                    EdgesToTest := EdgesToTest∪e
25:                end if
26:            end for
27:        end if
28:        TestedEdges := TestedEdges ∪ E
29:    end while
30:    return Cones
31: end function
```

The exploration of the neighboring edges corresponds to tilting the ray r in Algorithm 1, as in rotating a hyperplane in the gift wrapping method. One may wonder why the exploration of the edge skeleton in Algorithm 1 needs to continue after the statement on line 5. This is because the cone C has the potential to intersect many cones in P, particularly if P has small cones. Furthermore it is reasonable to wonder why we bother checking cone containment when computing the intersection of two cones provides more useful information. Checking cone containment means checking if each of the generators of C is contained in C_E, which is a far less computationally expensive operation than computing the intersection of two cones.

In the Newton-Puiseux algorithm to compute series expansions, we are interested only in the edges on the lower hull of the Newton polytope, i.e. those edges that have an upward pointing inner normal [17]. For Puiseux for space curves, the expansions are normalized so that the first exponent in the tropism is positive. Algorithm 2 is then adjusted so that calls to the edge skeleton computation of Algorithm 1 are made with rays that have a first component that is positive.

Algorithm 2 Finds pretropisms for a given tuple of polytopes $(P_1, P_2, \ldots, P_n)$.

```
1: function FindPretropisms(P_1, P_2, ..., P_n)
2:     Cones := set of normal cones to edges in P_1
3:     for i := 2 to n do
4:         NewCones := ∅
5:         for Cone in Cones do
6:             NewCones := NewCones ∪
                   ExploreEdgeSkeleton(P_i, Cone)
7:         end for
8:         Cones := NewCones
9:     end for
10:    Pretropisms := set of generating rays for each
         cone in Cones
11:    return Pretropisms
12: end function
```

3.1 Correctness

To see that this algorithm will do what it claims, we must define an additional term. A *pretropism graph* is the set of edges for a polytope that have normal cones intersecting a given cone. We will now justify why the cones output by Algorithm 1 correspond to the set of cones that live on a pretropism graph.

Theorem 1 *Pretropism graphs are connected graphs.*

Proof. Let C be a cone, and let P be a polytope with edges e_1, e_2 such that they are in the pretropism graph of C. Let C_1 be the cone of the intersection

of the normal cone of e_1 with C, and let C_2 be the cone of the intersection of the normal cone of e_2 with C. If we can show that there exists a path between e_1 and e_2 that remains in the pretropism graph, then the result will follow.

Let n_1 be a normal to e_1 that is also in C_1 and let n_2 be a normal to e_2 that is also in C_2. Set $n = tn_1 + (1-t)n_2$ where $0 \leq t \leq 1$. Consider varying t from 0 to 1; this creates the cone C_n, a cone which must lie within C, as both n_1 and n_2 lie in that cone. As n moves from 0 to 1, it will progressively intersect new faces of P that have all of their edges in the pretropism graph. Eventually, this process terminates when we reach e_2, and we have constructed a path from e_1 to e_2. Since a path always exists, we can conclude that pretropism graphs are connected graphs. ■

Since pretropism graphs are connected, Algorithm 1 will find all cones of edges on the pretropism graph. In Algorithm 2, we repeatedly explore the edge skeleton of polytope P_i, and use the pruned set of cones to explore P_{i+1}. From this, it is clear that Algorithm 2 will suffice to find all pretropisms.

4 Comparison With Our Previous Algorithm

Our algorithm in [15] restricted the pruning of the cone intersections in a vertical fashion: nodes in the tree with cone intersections that yield only $\{0\}$ will not have any children. That algorithm works well for polytopes with randomly generated coordinates.

In this paper we consider polytopes that are not in generic position. In this situation, intersecting normal cones to edges may lead to cones of normals of higher dimensional cones. At the same level in the tree we can then have duplicate cones or cones that are contained in other cones. In those cases were one cone is contained in another, the smaller cone can be pruned from the tree. We call this type of pruning horizontal pruning. For generic polytopes horizontal pruning would not reduce the number of cone intersections. However, in special cases like the cyclic n-root problem, there is the potential to dramatically reduce the scope of the problem through horizontal pruning.

To illustrate horizontal pruning, consider Figure 1. These graphs illustrate computing the pretropisms for three fictitious, non-generic polytopes P_1, P_2, P_3 with the two distinct algorithms. Nodes A, B, C represent the cones of the edges of P_1, the row below that represents the resulting cones from performing Algorithm 1 with P_2 and A, B, or C. The row below that varies in the two figures. In the tree at the top of Figure 1, the process iterates and Algorithm 1 is performed with P_3 and each of the input cones D through J. The tree at the bottom of Figure 1 shows how the horizontal pruning has the potential to improve over the previous algorithm. Since there are duplicate nodes for F, G, and H, each of these paths only needs to be followed once. Though this does not lead to dramatic improvements in this fictitious case, as the number of polytopes increases, the benefit of pruning compounds.

5 Computational Experiments

We developed a preliminary version of Algorithm 2 in Sage [16], using its modules for lattice polytopes [14], and polyhedral cones [7]; Sage uses PPL [4] to compute cone intersections. Our preliminary code is available at https://github.com/sommars/GiftWrap. We ran the code on a Red Hat Enterprise Linux workstation of Microway, with Intel Xeon E5-2670 processors at 2.6 GHz.

Instead of directly calculating the pretropisms of the Newton polytopes of the cyclic n-root problem, we chose to calculate pretropisms of the reduced cyclic n-root problem. This reformulation [8] is obtained by performing the substitution $x_i = \frac{y_i}{y_0}$ for $i = 0 \ldots n-1$. Clearing the denominator of each equation leaves the first $n-1$ equations as polynomials in $y_1, \ldots y_{n-1}$. We compute pretropisms of the Newton polytopes of these $n-1$ equations because they yield meaningful sets of pretropisms. Calculating with the reduced cyclic n-roots problem has the benefit of removing much of the symmetry present in the standard cyclic n-roots problem, as well as decreasing the ambient dimension by one. Unlike the standard cyclic n-roots problem, some of the polytopes of the reduced cyclic n-roots problem are full dimensional, which leads to calculation speed ups. A simple transformation can be performed on the pretropisms we calculate of reduced cyclic n-root problem to convert them to the pretropisms of cyclic n-root problem, so calculating the pretropisms of reduced cyclic n-roots problem is equivalent to calculating the pretropisms of the cyclic n-roots problem.

In Table 1, we have recorded the number of cone intersections performed and the number of times cone containments let us avoid performing additional intersections for each of the reduced cyclic n-root systems with $n \leq 10$. Table 1 also contains a comparison between the two sums, which acts as a way of evaluating the quality of the algorithms. We consider the unit of work of each algorithm to be the total number of intersections performed, as that is the constraining part of the algorithm. As n increases, the ratio tends to increase as well, demonstrating that Algorithm 2 represents a substantial improvement over our previous algorithm. We expect that the result would become even more dramatic with higher n, but in our testing, our previous algorithm was too inefficient to terminate for $n > 10$.

n	intersections	containments	sum	intersections	containments	sum	ratio
4	63	2	65	54	2	56	1.16071
5	750	20	770	395	5	400	1.92500
6	4,531	1,232	5,763	2,982	291	3,273	1.76076
7	105,982	5,767	111,749	18,798	343	19,141	5.83820
8	479,640	181,507	661,147	145,125	3,922	149,047	4.43582
9	9,232,384	1,993,049	11,225,433	1,101,563	16,313	1,117,876	10.04175
10	70,026,302	23,838,851	93,865,153	8,846,353	165,203	9,011,556	10.41608

Table 1: Columns two through four contain results when our previous algorithm is applied to the reduced cyclic n-roots problem while columns five through seven contain the results of Algorithm 2. The final column represents the ratio of the previous sum to the sum of Algorithm 2.

6 Comparison with Gfan

As we reported in [15], on randomly generated polytopes, our code was competitive with Gfan [12]. Although the additional pruning criteria presented in this paper are promising, on the specific cyclic n-roots problem, our Python prototype is not as good as the compiled code of Gfan. Our code tends to be slower by a factor of two, but we hope to be more competitive if we improve our ability to exploit the symmetry of the polytopes.

The computational complexity is such that high level parallelism is effective. Instead of iterating through all of the cones from line 5 of Algorithm 2, we can create a queue of them and then perform Algorithm 1. We then initialize some number of processes and give them successive cones from the queue until the queue is empty. Once the queue is empty, the resulting cones are pruned and combined and the algorithm iterates. We have incorporated this parallelism into our prototype Sage code.

References

[1] D. Adrovic and J. Verschelde. Computing Puiseux series for algebraic surfaces. In J. van der Hoeven and M. van Hoeij, editors, *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation (ISSAC 2012)*, pages 20–27. ACM, 2012.

[2] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8(3):295–313, 1992.

[3] J. Backelin. Square multiples n give infinitely many cyclic n-roots. Reports, Matematiska Institutionen 8, Stockholms universitet, 1989.

[4] R. Bagnara, P. Hill, and E. Zaffanella. The Parma Polyhedral Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[5] G. Björck and B. Saffari. New classes of finite unimodular sequences with unimodular Fourier transforms. Circulant Hadamard matrices with complex entries. *C. R. Acad. Sci. Paris, Série I*, 320:319–324, 1995.

[6] T. Bogart, A. Jensen, D. Speyer, B. Sturmfels, and R. Thomas. Computing tropical varieties. *Journal of Symbolic Computation*, 42(1):54–73, 2007.

[7] V. Braun and M. Hampton. `polyhedra` module of Sage. The Sage Development Team, 2011.

[8] I. Emiris. *Sparse Elimination and Applications in Kinematics*. PhD thesis, University of California at Berkeley, Berkeley, 1994.

[9] I. Emiris, V. Fisikopoulos, and B. Gärtner. Efficient edge-skeleton computation for polytopes defined by oracles. *Journal of Symbolic Computation*, 73:139–152, 2016.

[10] H. Führ and Z. Rzeszotnik. On biunimodular vectors for unitary matrices. *Linear Algebra and its Applications*, 484:86–129, 2015.

[11] K. Fukuda and A. Prodon. Double description method revisited. In *Selected papers from the 8th Franco-Japanese and 4th Franco-Chinese Conference on Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, 1996.

[12] A. Jensen. Computing Gröbner fans and tropical varieties in Gfan. In M. Stillman, N. Takayama, and J. Verschelde, editors, *Software for Algebraic Geometry*, volume 148 of *The IMA Volumes in Mathematics and its Applications*, pages 33–46. Springer-Verlag, 2008.

[13] D. Maclagan and B. Sturmfels. *Introduction to Tropical Geometry*, volume 161 of *Graduate Studies in Mathematics*. American Mathematical Society, 2015.

[14] A. Novoseltsev. `lattice_polytope` module of Sage. The Sage Development Team, 2011.

[15] J. Sommars and J. Verschelde. Exact gift wrapping to prune the tree of edges of Newton polytopes to compute pretropisms. `arXiv:1512.01594`.

[16] W. Stein et al. *Sage Mathematics Software (Version 6.9)*. The Sage Development Team, 2015. `http://www.sagemath.org`.

[17] R. Walker. *Algebraic Curves*. Princeton University Press, 1950.

[18] G. Ziegler. *Lectures on Polytopes*. Springer-Verlag, 1995.

Computing the Fréchet Distance between Real-Valued Surfaces

Kevin Buchin * Tim Ophelders * Bettina Speckmann *

1 Introduction

The problem of measuring the similarity between shapes has recently gained much attention. While many measures have been defined, algorithms to compute such measures have been found for only some of them. We consider the problem of comparing real-valued functions $f\colon M \to \mathbb{R}$ on surfaces, focusing in particular on spheres and disks of constant boundary, i.e., $f(x) = f(x')$ for all $x, x' \in \partial M$. The kind of similarity we investigate is that under continuous deformations of surfaces, such as in Figure 1. Here shapes that can be deformed into each other have distance 0, otherwise, shapes have some meaningful positive distance. Two natural computational problems arise for each measure, namely deciding whether two images have distance 0, and the more general problem of computing the distance between two images.

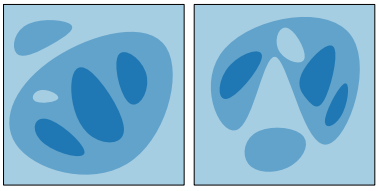


Figure 1: Pictures that can deform into each other.

Major applications of computing such measures are in the comparison of medical imagery. For example, when comparing two MRI or CT scans of lungs, the images are often not aligned due to breathing and gravity. It is important to align the images through deformation to locate differences.

Definitions, background and results. Given two functions $f\colon M \to \mathbb{R}^k$ and $g\colon M \to \mathbb{R}^k$ with common parameter space M, their *Fréchet distance* is defined by Equation 1, where $\mu\colon M \to M$ ranges over orientation preserving homeomorphisms and $d(\cdot,\cdot)$ is the underlying norm of $\mathbb{R}^k$. Essentially, the Fréchet distance captures the similarity between two functions by realigning their parameter spaces to minimize the maximum difference in function value of aligned points. We assume that f and g are piecewise-linear.

$$d_F(f,g) = \inf_{\mu\colon M\to M} \sup_{x\in M} d(f(x), g\circ\mu(x)). \quad (1)$$

*Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands,
[k.a.buchin|t.a.e.ophelders|b.speckmann]@tue.nl

Efficient algorithms for computing $d_F(f,g)$ exist for L^p norms if f and g are polylines [3], so if $M = [0,1]$ or $M = S^1$ for closed polylines. The computational complexity in the case that f and $g\colon M \to \mathbb{R}^k$ are (triangulated) surfaces is much less understood. The problem is known to be NP-hard [7] also when $k = 2$ [4, 5]. But it is not known whether it is actually in NP, in fact it is only known to be upper semi-computable for surfaces in $\mathbb{R}^k$ [2].

We show that the problem is in NP for $k = 1$ if M is a topological sphere or disk with constant boundary. Additionally, we show that even for $k = 1$, computing a factor $2 - \varepsilon$ approximation of the Fréchet distance is NP-hard. We achieve our results on surfaces (Section 3) by first defining a suitable similarity measure between contour trees, which we show to be NP-complete to approximate as well (Section 2).

In previous work, a few variants on the comparison of surfaces under the Fréchet distance have been investigated. For instance, there are efficient algorithms for computing the Fréchet distance with certain constraints on the homeomorphisms μ [5] and for computing the *weak Fréchet distance* [2] between triangulated surfaces homeomorphic to the disk.

2 Contour tree distance

The Reeb graph [8] of a function $f\colon M \to \mathbb{R}$ is the quotient space $M/\!\sim_f$ (endowed with the quotient topology) where $a \sim_f b$ if and only if a and b are in the same connected component of the level set $f^{-1}(f(a))$. Denote by $\mathcal{R}_f$ the corresponding quotient map. Because f associates a single real number to each equivalence class of $\sim_f$, the resulting Reeb graph has a natural $\mathbb{R}$-valued function associated with it, namely the (unique) function $f'\colon M/\!\sim_f \to \mathbb{R}$ satisfying $f' \circ \mathcal{R}_f = f$. If M is the disk or the 2-sphere, the Reeb graph forms a tree called a contour tree.

For the sake of compact representation, in this paper we assume each surface to be triangulated to form a simplicial 2-complex. Furthermore, we assume function values along edges of Reeb graphs to be linearly interpolated between the values of the vertices at their endpoints. In this representation, the contour tree of a surface with n faces has complexity $O(n)$ and can be computed in $O(n \log n)$ time [11].

Based on the Fréchet distance between f and g, we derive a computationally simpler measure that abstracts from the realizability of the matching μ be-

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

tween spheres or disks. Throughout this paper, we use the notation $\mathbb{X} = M/\sim_f$ and $\mathbb{Y} = M/\sim_g$ for the contour trees of f and g, respectively. We shall denote the vertex set of $\mathbb{X}$ by $V(\mathbb{X})$ (that is, the saddle points and minima and maxima of f) and its edge set by $E(\mathbb{X})$. With slight abuse of notation, we reuse function names f and g for the natural $\mathbb{R}$-valued functions associated with the contour trees $\mathbb{X}$ and $\mathbb{Y}$. Our distance measure d_C compares the contour trees $\mathbb{X}$ and $\mathbb{Y}$ of f and g. We define the contour tree distance d_C between $f\colon \mathbb{X} \to \mathbb{R}$ and $g\colon \mathbb{Y} \to \mathbb{R}$ as

$$d_C(f,g) = \inf_{\tau \in \mathcal{M}(\mathbb{X},\mathbb{Y})} \sup_{(x,y)\in\tau} |f(x) - g(y)|,$$

where $\tau \subseteq \mathbb{X} \times \mathbb{Y}$ is drawn from a specific class of matchings $\mathcal{M}(\mathbb{X},\mathbb{Y})$, defined below. So τ defines a correspondence between contour trees, such that $(x,y) \in \tau$ if some points on contours x and y are matched by a corresponding matching μ on M. Denote $\tau(x) = \{y \mid (x,y) \in \tau\}$ and $\tau^{-1}(y) = \{x \mid (x,y) \in \tau\}$. The class $\mathcal{M}(\mathbb{X},\mathbb{Y})$ captures the essential (but not all) properties of an orientation preserving matching μ. We define $\mathcal{M}(\mathbb{X},\mathbb{Y})$ as the set of matchings τ for which the following properties hold:

1. τ is a connected subset of $\mathbb{X} \times \mathbb{Y}$;
2. $\tau(x)$ is a nonempty subtree of $\mathbb{Y}$ for each $x\colon \mathbb{X}$;
3. $\tau^{-1}(y)$ is a nonempty subtree of $\mathbb{X}$ for each $y\colon \mathbb{Y}$.

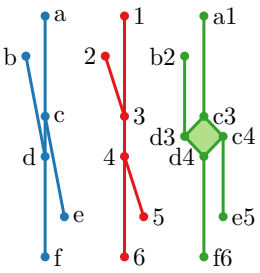


Figure 2: Two trees (left) and a matching.

Here, the term *subtree* is used to denote a connected subset of a tree, not necessarily containing leaves of that tree. By Conditions 2 and 3, each connected set matches with a connected set, and Condition 1 ensures continuity.

Figure 2 shows an example of a matching between trees. The two-dimensional patch of this matching represents a many-to-many correspondence. For a matching $\mu\colon M \to M$ between surfaces f and g, define $T(\mu)$ to be the corresponding matching between the Reeb graphs of f and g:

$$T(\mu) = \{(\mathcal{R}_f(x), \mathcal{R}_g \circ \mu(x)) \mid x \in M\} \qquad (2)$$

Lemma 1 *If $\mu\colon M \to M$ is orientation preserving and $\tau = T(\mu)$, then $\tau \in \mathcal{M}(\mathbb{X},\mathbb{Y})$.*

Proof. Consider such a matching μ and the correspondence τ between $\mathbb{X}$ and $\mathbb{Y}$. We show all three conditions on τ hold. The set $\{(x,y) \mid \mu(x) = y\}$ is a connected subset of $S^2 \times S^2$, and hence its image under the quotient map $(x,y) \mapsto (\mathcal{R}_f(x), \mathcal{R}_g(y))$ is connected, so Condition 1 holds. Because μ preserves orientation, $\tau(x)$ is connected, and by surjectivity nonempty. Hence, $\tau(x)$ and symmetrically $\tau^{-1}(y)$ is a nonempty subtree of $\mathbb{Y}$ and $\mathbb{X}$, respectively. □

Corollary 2 $d_C(f,g) \le d_F(f,g)$.

Lemma 3 *Computing $d_C(f,g)$ is in NP.*

By Lemma 1 we have for each orientation preserving homeomorphism μ, that the matching $\tau = T(\mu)$ satisfies $\tau \in \mathcal{M}(\mathbb{X},\mathbb{Y})$. Hence, $d_C(f,g) \le d_F(f,g)$. On the other hand, a matching $\tau \in \mathcal{M}(\mathbb{X},\mathbb{Y})$ does not need to correspond to an orientation preserving homeomorphism on M, as illustrated in Section 3.1.

To test whether the contour tree distance between two trees is zero, one needs to test only whether the trees are equal. We represent trees canonically by exhaustively removing degree 2 vertices that lie on the segment connecting the two adjacent vertices, and replacing them by a single edge between those vertices. This reduces the problem to labeled unordered unrooted tree isomorphism, solvable in linear time [1].

Computing the contour tree distance and Fréchet distance between trees are different problems. In fact, one major limitation of the Fréchet distance for trees is that non-homeomorphic trees have infinite Fréchet distance. Nonetheless, algorithms for computing the Fréchet distance between trees have been investigated before, yielding an $O(n^{5/2})$ time algorithm [4].

2.1 NP-hardness

We show that approximating the contour tree distance between $\mathbb{R}$-valued trees within factor 2 is NP-hard by a reduction from the NP-hard problem Exact Cover by 3-sets [6].

Definition 1 Exact Cover by 3-sets (X3C)
Input: A set $\mathcal{S}$ of m subsets of $\{1,\dots,k\}$ of size 3.
Output: Is there a subset of $\mathcal{S}$ consisting of $k/3$ disjoint triples whose union is $\{1,\dots,k\}$?

We introduce the gadgets used in our reduction in Figure 3. Gadget Y^* is a long segment from position 0 to position $6k+6$. Gadget Y_l ($l \in \{1,\dots,k\}$) is a path from position 1 to $6k+6$ with a single zig-zag of radius 2 around position $6l$. Similarly, gadget $X_{i,j}$ ($i \in \{1,\dots,m\}$, $j \in \{1,2,3\}$) has a single zig-zag around position $6 \cdot s(i,j)$, but with radius 1. The function s aligns the center of the zig-zag of $X_{i,j}$ with that of $Y_{s(i,j)}$, such that gadget $X_{i,j}$ has a contour tree

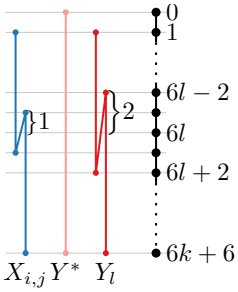


Figure 3: Gadgets.

distance of 1 to Y^* and $Y_{s(i,j)}$, but a contour tree distance of 2 to any gadget Y_l with $l \neq s(i,j)$.

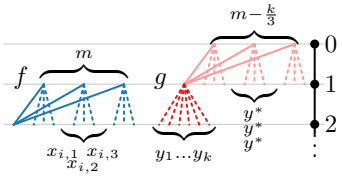


Figure 4: Connecting gadgets into trees f and g.

The function s can be configured such that each triple of gadgets $(X_{i,1}, X_{i,2}, X_{i,3})$ corresponds to one of the m subsets of $\mathcal{S}$. We connect the three elements of each triple at a common vertex at position 1, and finally connect all triples at a common vertex at position 2 (blue in Figure 4) to form tree $f\colon \mathbb{X} \to \mathbb{R}$.

Similarly, all gadgets Y_l correspond to an element of $\{1, \ldots, k\}$, and all Y_l are connected to a common vertex at position 1. To obtain a low contour tree distance, $k/3$ triples of f must match the Y_l gadgets exactly; then what remains in f are $m - k/3$ triples that must be matched elsewhere. Each such unmatched triple of f is then forced to match with three copies of Y^*, connected at a vertex at position 0 to form a so called Y^*-triple. We use $m - k/3$ such Y^*-triples, each connected to the Y_l gadgets at position 1 to form tree $g\colon \mathbb{Y} \to \mathbb{R}$. We use a solution to X3C to derive a matching using only many-to-one correspondences between f and g, even though $\mathcal{M}(\mathbb{X}, \mathbb{Y})$ also permits many-to-many correspondences. In the full paper we also show that any many-to-one matching can be realized as an orientation preserving homeomorphism on the sphere, such that computing the Fréchet distance between $\mathbb{R}$-valued spheres is NP-hard, see Theorem 5.

Theorem 4 *Computing a $(2-\varepsilon)$-approximation of the contour tree distance is NP-complete.*

Theorem 5 *Computing a $(2-\varepsilon)$-approximation of the Fréchet distance of $\mathbb{R}$-valued spheres is NP-hard.*

3 Surfaces

We consider two surfaces: the disk $M = [0,1]^2$ and the sphere S^2. Not all matchings between the contour trees $\mathbb{X}$ and $\mathbb{Y}$ can be realized as orientation preserving homeomorphisms on the sphere as illustrated in Section 3.1. In the case of the disk, the boundaries must also be matched, which imposes additional constraints on the matching of the interiors. We prove that the Fréchet distance between $\mathbb{R}$-valued spheres or disks with constant boundary is in NP in Section 3.2. For this we use properties of Euler diagrams.

3.1 An unrepresentable matching

Consider the two rooted trees $\mathbb{X}$ and $\mathbb{Y}$ of Figure 5. The leaves of $\mathbb{X}$ are labeled $x_{i,j}$ ($i \in \{1, \ldots, 6\}$, $j \in \{1,2,3\}$) and the leaves of $\mathbb{Y}$ are labeled $y_{k,l}$ ($k \in \{1, \ldots, 9\}$, $l \in \{1,2\}$). For both trees, leaves with the same i or k are grouped in subtrees. Based on the complete bipartite graph $K_{3,3}$ with vertices $v_1, \ldots, v_6$ and edges $e_1, \ldots, e_9$, we construct a matching τ between those subtrees as follows. For an edge $e_k = (v_i, v_{i'})$ of $K_{3,3}$, match the path from $y_{k,1}$ to $y_{k,2}$ with the path between unused vertices $x_{i,j}$ and $x_{i',j'}$. Match the edge from the root to group i of $\mathbb{X}$ with the edges of $\mathbb{Y}$ from the root to the three groups that match with $x_{i,1}$, $x_{i,2}$ and $x_{i,3}$. Then $\tau \in \mathcal{M}(\mathbb{X}, \mathbb{Y})$ does not match any path from $y_{k,1}$ to $y_{k,2}$ (of edge $e_k = (v_i, v_{i'})$) with any group of $\mathbb{X}$ not containing any $x_{i,j}$ or $x_{i',j'}$. However, because $K_{3,3}$ is not a planar graph, this matching cannot be realized on the sphere, as illustrated in Figure 5.

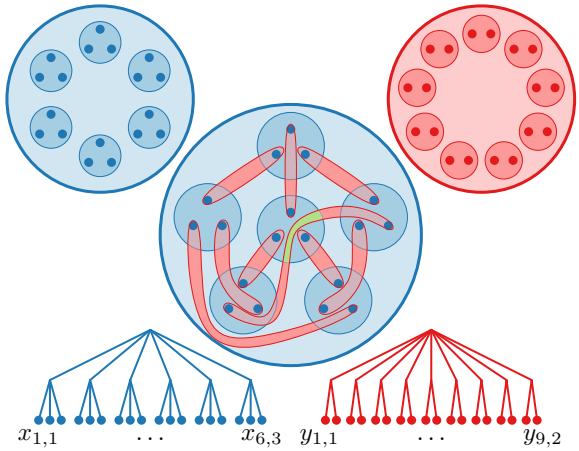


Figure 5: Top: surfaces. Bottom: trees $\mathbb{X}$ and $\mathbb{Y}$. Middle: a matching in which a subtree of $\mathbb{Y}$ must intersect an additional subtree of $\mathbb{X}$.

3.2 Fréchet distance in NP

An Euler diagram is a set of topological disks, drawn in the plane to capture relations such as overlap or containment between them. There are eight such relations: disjoint, equal, inside, contains, covered, cover, meet, overlap [10]. For a set D of n disks and a relation $\mathcal{P}(a,b)$ between each pair $a, b \in D$ of those disks, the tuple $(D, \mathcal{P})$ is called a *topological expression*. It was shown by Schaefer, Sedgwick and Štefankovič [9] that it is in NP to decide whether D can be drawn in the plane to satisfy all relations of $\mathcal{P}$.

We show that deciding whether the Fréchet distance between $\mathbb{R}$-valued surfaces f and $g\colon M \to \mathbb{R}$ is at most ε is in NP if M is a sphere. First consider the case where $M = [0,1]^2$ is a topological disk

where f and g have constant boundary: $f(p) = f(q)$ and $g(p) = g(q)$ for all $p, q \in \partial M$.

We can represent the contour trees $\mathbb{X}$ and $\mathbb{Y}$ as rooted trees, and represent each subtree as a disk. A vertex is represented by a disk with punctures (one per edge to a subtree), and an edge e is represented by two nested disks, whose difference is an annulus $A(e)$. Let $W(\cdot)$ denote the disks of a tree.

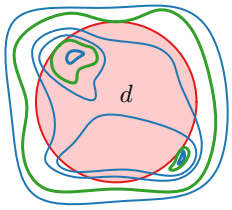


Figure 6: $\mathbb{X}'$ (green) refining $\mathbb{X}$ (blue) for a single $d \in W(\mathbb{Y})$.

Define matching $\mu\colon M \to M$ to be an *ε-matching* if and only if $|f(x) - g(\mu(x))| \leq \varepsilon$ for all x. Consider any ε-matching μ between $\mathbb{X}$ and $\mathbb{Y}$. A polynomial amount of information about μ is used to derive a topological expression that captures the relations between the disks (of $\mathbb{X}$ and $\mathbb{Y}$) as drawn by μ. We construct trees $\mathbb{X}'$ and $\mathbb{Y}'$ that have additional vertices along the edges. We include for each disk d of $\mathbb{X}$ the extremes p on $\mathbb{Y}$ of the image of the boundary of d as vertices of $\mathbb{Y}'$. That is, $p \in \partial\mathcal{R}_g(\mu(\partial d)) \subseteq V(\mathbb{Y}')$, and we require the boundary of these disks to touch but not intersect ∂d. Symmetrically, refine $\mathbb{X}$ into $\mathbb{X}'$ and let D be the disks of $W(\mathbb{X}') \supseteq W(\mathbb{X})$, and $W(\mathbb{Y}') \supseteq W(\mathbb{Y})$, see Figure 6.

Consider any drawing of D that satisfies the constraints imposed by the topological expression derived from μ. We claim that if μ was an ε-matching, we can parameterize f and g such that they form an ε-matching μ'. Where boundaries of disks of $\mathbb{X}'$ and $\mathbb{Y}'$ intersect, both function values are fixed and differ by at most ε. Consider any face F in the drawing $\bigcup_{d\in D} \partial d$ of the boundaries of disks. It is bounded by (a subset of) boundaries of either a vertex or edge of $\mathbb{X}'$ and those of a vertex or edge of $\mathbb{Y}'$. By construction, F is either entirely contained in the image of a vertex, or disk boundaries of different surfaces intersect on ∂F. In the former case, function values are fixed for ∂F and differ by at most ε. In the latter case, linearly interpolate the function values between those at intersections. The function value (of f and g) on each boundary of F is then either constant, or (only if F lies in the intersection of two annuli and has genus 0) two linear interpolations (back and forth) between the values at the boundary of an annulus.

If either function is constant along ∂F, assign the same constant to the interior, such that the other function can by interpolated arbitrarily. If neither function is constant, $F \subseteq A(e) \cap A(e')$ lies in the intersection of annuli ($e \in E(\mathbb{X}')$ and $e' \in E(\mathbb{Y}')$). If F is homeomorphic to an annulus, we interpolate f and g in its interior linearly between the boundaries. In the final case where both f and g are interpolated along the boundary of F, we separate the regions where the interpolation of f occurs from that of g, such that the function values at any point differ by at most ε.

Deciding whether the topological expression (constructed using a polynomial amount of information about μ) has a realization in the plane is decidable in nondeterministic polynomial time. For any ε-matching μ, this topological expression has a solution, and each such solution encodes an ε-matching. Theorems 6 and 7 follow.

Theorem 6 *Deciding whether $d_F(f, g) \leq \varepsilon$ for disks f and g with constant boundary is in NP.*

Theorem 7 *Deciding whether $d_F(f, g) \leq \varepsilon$ for spheres f and g is in NP.*

Acknowledgements. K. Buchin, T. Ophelders, and B. Speckmann are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.207 (K. Buchin) and no. 639.023.208 (T. Ophelders & B. Speckmann).

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] H. Alt and M. Buchin. Can we compute the similarity between surfaces? *DCG*, 43(1):78–99, 2010.

[3] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *IJCGA*, 5(01n02):75–91, 1995.

[4] K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: Some simple hard cases. In *Proc. 18th ESA*, pages 63–74, 2010.

[5] K. Buchin, T. Ophelders, and B. Speckmann. Computing the similarity between moving curves. In *Proc. 23rd ESA*, pages 928–940, 2015.

[6] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. *WH Freeman & Co., San Francisco*, 1979.

[7] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. PhD thesis, Freie Universität Berlin, Germany, 1998.

[8] G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *C. R. de L'Académie des Sciences*, 222:847–849, 1946.

[9] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. *Journal of Computer and System Sciences*, 67(2):365 – 380, 2003.

[10] M. Schaefer and D. Štefankovič. Decidability of string graphs. In *Proc. 33rd STOC*, pages 241–246, 2001.

[11] M. Van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. 13th SoCG*, pages 212–220, 1997.

Discrete Fréchet Distance for Uncertain Points

Maike Buchin* Stef Sijben*

Abstract

We consider the problem of computing the discrete Fréchet distance between polygonal curves with uncertain points, i.e. the coordinates of the vertices are not known exactly, but are given by a probability distribution. In this case, the discrete Fréchet distance is a random variable. We show that the distribution function for a given coupling can be efficiently evaluated and give an algorithm to compute the coupling with maximum probability of realizing a given discrete Fréchet distance.

1 Introduction

The discrete Fréchet distance is a popular measure for the similarity of two polygonal curves with many applications. In the standard case where the locations of the vertices are known exactly, it can be computed in $O(n^2)$ time for two curves of length at most n [5]. Recently, Agarwal et al. discovered a subquadratic time algorithm for this problem [1].

In practice, the curves are often based on trajectories collected from a moving entity, e.g. using a GPS tracking device. These devices do not provide precise locations, but rather an estimate of the location, typically including an error margin. If the goal is to compute the discrete Fréchet distance, a single outlying observation may lead to a very different coupling than the distance based on real locations, as is illustrated in Figure 1. One proposed solution is the (discrete) Fréchet distance with shortcuts [4, 3], which can remove outliers from one of the curves. Here we will follow the approach of including the uncertainty in the model.

Several models have been proposed to incorporate uncertainty in geometric problems [7]: In the *imprecise points* model each point lies in a given region. With *indecisive points*, each point is selected from a finite set of candidate locations. For *uncertain points*, the location of a point is described using a probability distribution based on the observed location. The uncertain points model is the most general, and it is closest to the practical applications, where a point is likely to be close to the observed location, but large errors are possible. The most commonly used distribution is a circular normal distribution with the mean

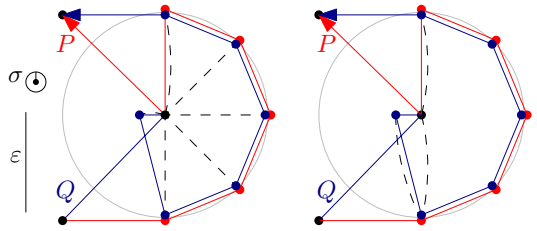


Figure 1: Example of two trajectories (slightly shifted for visual clarity) with discrete Fréchet distance ε. With precise points, all points on the circle are matched to a point in the middle (left). With uncertain points (all normally distributed with standard deviation σ), the optimal coupling matches most vertices to one centered at the same location (right).

at the observed location. The variance can be given by the tracking device's estimated error or based on other measurements.

In the case of imprecise points, where the points are known to be in a given region, efficient algorithms exist to compute the smallest possible discrete Fréchet distance if the regions are d-dimensional balls or for axis-parallel boxes under the L_∞ norm [2]. Computing an upper bound for the discrete Fréchet distance is NP-hard in the imprecise setting [6].

2 Preliminaries

Formally, a polygonal curve with uncertain points P is a sequence of vertices $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n$. $\boldsymbol{P}_i$ is a random point in $\mathbb{R}^d$ distributed according to a certain (known) distribution. For example, a vertex may be obtained by a GPS fix at a certain time and be normally distributed around the observed location with a certain variance: $\boldsymbol{P}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma_i^2)$. We assume that all $\boldsymbol{P}_i$ are independent.

Consider two polygonal curves with uncertain points P and Q of length n and m respectively and assume w.l.o.g. that $n \geq m$. A *coupling* C between P and Q is a sequence of pairs $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$ such that $a_1 = b_1 = 1$, $a_k = n$, $b_k = m$ and for each $i \in \{1, \ldots, k-1\}$ one of the following holds:

- $a_{i+1} = a_i$ and $b_{i+1} = b_i + 1$,
- $a_{i+1} = a_i + 1$ and $b_{i+1} = b_i$, or
- $a_{i+1} = a_i + 1$ and $b_{i+1} = b_i + 1$.

*Department of Mathematics, Ruhr-Universität Bochum, {Maike.Buchin,Stef.Sijben}@rub.de

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

The discrete Fréchet distance of polygonal curves P and Q is defined as

$$d_{dF}(P,Q) = \min_{C} \max_{i\in\{1,\ldots,k\}} |\boldsymbol{P}_{a_i} - \boldsymbol{Q}_{b_i}|,$$

where C ranges over all couplings between P and Q.

The discrete Fréchet distance is usually computed using the free space matrix: A table of size $n \cdot m$, where each cell (i,j) represents a pair of points $\boldsymbol{p}_i, \boldsymbol{q}_j$ from P and Q, respectively. This cell is free if $|\boldsymbol{p}_i - \boldsymbol{q}_j| \leq \varepsilon$. Then, $d_{dF}(P,Q) \leq \varepsilon$ if and only if there is a bimonotone path from $(1,1)$ to (n,m) visiting only free cells.

If the distribution has unbounded support, there are no upper or lower bounds on the value of the discrete Fréchet distance (other than the trivial lower bound of 0). Instead, its value is given by a probability distribution. Ideally, one would like to be able to compute properties of this distribution, e.g.

- distribution function $F(\varepsilon) = \mathbb{P}\left[d_{dF}(P,Q) \leq \varepsilon\right]$,
- probability density $f(\varepsilon) = \frac{d}{d\varepsilon}F(\varepsilon)$,
- quantiles $F^{-1}(\rho) = \inf\{\varepsilon \in \mathbb{R}^{\geq 0} \mid \rho \leq F(\varepsilon)\}$.

Note that if an algorithm for the distribution function is known, the others can be approximated using standard numerical techniques.

Figure 1 shows a pair of trajectories where uncertain points lead to a different result than precise points, in the sense that the coupling with the highest probability of achieving discrete Fréchet distance $\leq \varepsilon$ is different from the coupling with precise points at distance ε. For noisy data, the coupling produced using uncertain points seems more reasonable. In practice, e.g. when studying trajectories with GPS error, one is often not so much interested in the exact value of ε, but in finding a reasonable value for ε with a coupling that provides a good match between the two trajectories. In some cases, using uncertain points avoids intuitively unappealing couplings in favour of a more reasonable one that has a slightly larger distance.

One option to compute $F(\varepsilon)$ is to fix the position of each point, test whether $d_{dF}(P,Q) \leq \varepsilon$ for these locations and integrate each point over $\mathbb{R}^d$, weighted by the probability density of the point, i.e.:

$$F(\varepsilon) = \int_{\mathbb{R}^d} f_{\boldsymbol{P}_1}(\boldsymbol{p}_1) \ldots \int_{\mathbb{R}^d} f_{\boldsymbol{P}_n}(\boldsymbol{p}_n) \int_{\mathbb{R}^d} f_{\boldsymbol{Q}_1}(\boldsymbol{q}_1) \ldots \int_{\mathbb{R}^d} f_{\boldsymbol{Q}_m}(\boldsymbol{q}_m) I(\varepsilon) \, d\boldsymbol{q}_m \ldots d\boldsymbol{q}_1 d\boldsymbol{p}_n \ldots d\boldsymbol{p}_1,$$

where $I(\varepsilon)$ is the indicator function which is 1 if $d_{dF}(P,Q) \leq \varepsilon$ for the given coordinates and 0 otherwise.

However, evaluating these $d(n+m)$ nested integrals to a reasonable precision requires time exponential in the dimension [8], so other approaches are needed.

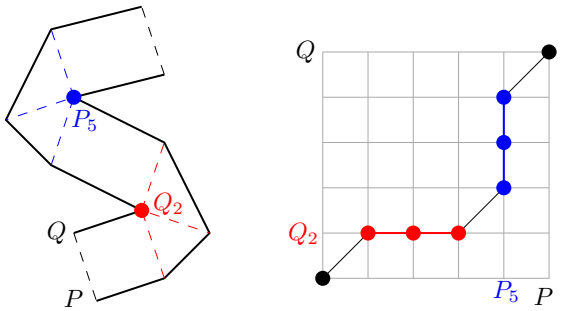


Figure 2: Two trajectories with a coupling and the corresponding free space matrix. The distribution functions for the red and blue segments, as well as each of the black points, can be computed independently.

3 Evaluating $F(\varepsilon)$ for a fixed coupling

We first consider how to evaluate $F_C(\varepsilon)$, i.e. the probability that a given coupling C realizes $d_{dF}(P,Q) \leq \varepsilon$. C can be represented by a bimonotone path through the free space matrix. We assume that this path makes no 90° turns. Any path can be converted to this form in linear time by diagonally going from the cell before the turn to the cell after the turn. This transformation can only increase $F_C(\varepsilon)$, since it removes some diagonals in the coupling and does not introduce any new ones.

A diagonal move from (i,j) to $(i+1,j+1)$ in the path breaks the trajectories up into two subtrajectories each such that

$$d_{dF}(P,Q) = \max\{d_{dF}(P[1\ldots i], Q[1\ldots j]), d_{dF}(P[i+1\ldots n], Q[j+1\ldots m])\}.$$

Since these are disjoint subtrajectories, the distributions of their discrete Fréchet distances are independent. This property allows us to break the path up into (possibly degenerate) horizontal and vertical segments which can be treated independently.

Let $C_1, \ldots, C_k$ be the segments of the coupling defined above and let $F_{C_\ell}(\varepsilon)$ be the probability that the segment C_ℓ realizes a discrete Fréchet distance between the induced subtrajectories of at most ε. We use independence to obtain $F_C(\varepsilon)$:

$$F_C(\varepsilon) = \prod_{\ell=1}^{k} F_{C_\ell}(\varepsilon).$$

As illustrated in Figure 2, there are three possible cases for $F_{C_\ell}(\varepsilon)$, which are easy to deal with:

1. The segment contains only a single point. This represents one point on each trajectory being matched to each other, thus the question reduces to $F_{C_\ell}(\varepsilon) = \mathbb{P}\left[|\boldsymbol{P}_i - \boldsymbol{Q}_j| \leq \varepsilon\right]$.

For normally distributed points the distance is related to the noncentral chi-squared distribution and this can be computed by evaluating its distribution function once.

2. A vertical segment represents a single point $\boldsymbol{P}_i$ being matched to several points $Q[j \dots j']$. Here, we fix $\boldsymbol{P}_i$, compute the probability that all other points are close enough to the fixed point and integrate $\boldsymbol{P}_i$ over all possible locations:

$$F_{C_\ell}(\varepsilon) = \int_{\mathbb{R}^d} f_{\boldsymbol{P}_i}(\boldsymbol{x}) \prod_{k=j}^{j'} \mathbb{P}\left[|\boldsymbol{Q}_k - \boldsymbol{x}| \le \varepsilon\right] d\boldsymbol{x}. \tag{1}$$

Evaluating these d nested integrals takes $O(c^d \cdot (j' - j))$ time, where c depends on the number of integration steps required, i.e. the integration technique used, the desired precision and details of the functions being integrated [8].

Again, for normally distributed points the probability can be computed using the distribution function of the noncentral chi-squared distribution.

3. A horizontal segment can be processed symmetrically to case 2.

All $F_{C_\ell}(\varepsilon)$ and hence $F_C(\varepsilon)$ can be computed in $O(c^d \cdot (n+m))$ time, where c again depends on the number of integration steps in cases 2 and 3.

4 Computing the optimal coupling

In this section, we present a dynamic programming algorithm that computes an optimal coupling for curves P and Q and distance ε, that is a coupling C for which $F_C(\varepsilon)$ is maximal, i.e. has the highest probability of achieving discrete Fréchet distance at most ε. The probability reached by this coupling is a lower bound for the distribution function $F(\varepsilon)$. Observe that the optimal coupling is one of the form described before, i.e. without 90° turns in the free space.

We use a table similar to the free space matrix, in which each cell represents a prefix of each trajectory and a path from the lower left corner to the cell represents a partial coupling. Using the independence of path segments separated by a diagonal move, we can decompose the optimal path to cell (i,j) into a final horizontal or vertical segment C_k from (i',j') to (i,j) (with $i' \le i$, $j' \le j$ and either $i' = i$ or $j' = j$), and an optimal path to $(i'-1, j'-1)$. These paths are then connected using a diagonal edge.

Let $p(i,j)$ denote the probability that the optimal coupling ending at (i,j) realizes $d_{dF}(\tau_1[1\dots i], \tau_2[1\dots j]) \le \varepsilon$ and let $\pi(i,j)$ be the coordinates (i',j') where this final segment starts. If the final segment is vertical and (i',j') is known, the probability is given by

$$\begin{aligned} p_v(i,j) &= p(i'-1, j'-1) \cdot F_{C_k}(\varepsilon) \\ &= p(i'-1, j'-1) \cdot \mathbb{P}\left[\varepsilon \ge \max_{k \in \{j', \dots, j\}} |\boldsymbol{P}_i - \boldsymbol{Q}_k|\right] \\ &= p(i'-1, j'-1) \\ &\qquad \cdot \int_{\mathbb{R}^d} f_{\boldsymbol{P}_i}(\boldsymbol{x}) \prod_{k=j'}^{j} \mathbb{P}\left[|\boldsymbol{Q}_k - \boldsymbol{x}| \le \varepsilon\right] d\boldsymbol{x}. \end{aligned}$$

If the final segment is horizontal, a similar expression exists for $p_h(i,j)$ and $p(i,j) = \max\{p_v(i,j), p_h(i,j)\}$.

To find (i',j') we search all possible predecessors, i.e. all cells in the jth row or ith column preceding (i,j), including (i,j) itself, select the (i',j') that maximizes the probability and set $p(i,j)$ and $\pi(i,j)$ accordingly. Then we construct the optimal coupling by following the $\pi(i,j)$ pointers back from (n,m).

The table contains $O(n^2)$ cells, for each cell we need to test $O(n)$ predecessors and computing $p(i,j)$ for a fixed predecessor takes $O(c^d \cdot (n))$ time to evaluate the integral as discussed before.

Theorem 1 *Given two curves with uncertain points P and Q of length n and a threshold ε, the coupling that has maximum probability of realizing $d_{dF}(P,Q) \le \varepsilon$ can be computed in time $O(c^d n^4)$, where c depends on the number of integration steps.*

Instead of a single coupling, the best k couplings can be computed by replacing $p(i,j)$ and $\pi(i,j)$ by lists of length k. The running time increases by a factor k.

Note that for fixed (i,j), $F_{C_k}(\varepsilon)$ is an increasing function in i' and j', since fewer points are matched to the fixed point as the segment becomes shorter. Thus, better running times are achieved in practice by searching predecessors backward from (i,j), stopping the search when $F_{C_k}(\varepsilon)$ for some (i',j') becomes smaller than the best known lower bound for $p(i,j)$.

Instead of computing the optimal coupling for a fixed distance ε, given a fixed probability ρ we can compute the coupling C that realizes $F_C(\varepsilon) \ge \rho$ for the smallest value of ε among all couplings, by searching over the distance values, using the dynamic programming algorithm in each step.

5 Experiments

The algorithms described in the previous sections were implemented in R and evaluated for several inputs. For the trajectories and ε shown in Figure 1, these experiments confirm that the coupling shown on the right indeed has a much higher probability (0.14) of realizing $d_{dF}(P,Q) \le \varepsilon$ than the coupling used to realize the discrete Fréchet distance for precise points

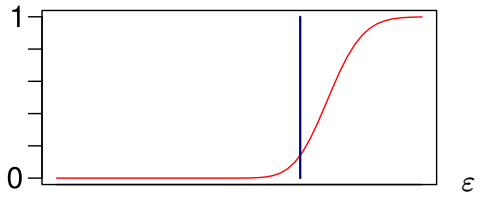


Figure 3: Probability that optimal coupling realizes $d_{dF}(P,Q) \leq \varepsilon$ for the trajectories in Figure 1. The discrete Fréchet distance for precise points is indicated by the blue line.

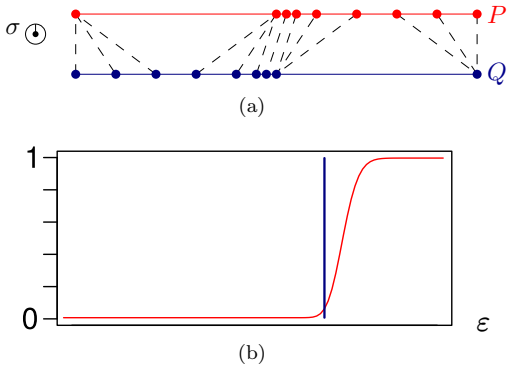


Figure 4: (a) Two curves with the coupling that realizes the minimal $d_{dF}(P,Q)$. Points are normally distributed with standard deviation σ. (b) Probability that the optimal curve realizes $d_{dF}(P,Q) \leq \varepsilon$. The discrete Fréchet distance for precise points is indicated by the blue line.

(0.00057). The probability that the optimal coupling realizes $d_{dF}(P,Q) \leq \varepsilon$ is plotted against ε in Figure 3. The discrete Fréchet distance with uncertain points tends toward slightly larger values than with precise points. This is to be expected, since the discrete Fréchet distance is a bottleneck distance and a single outlying point can cause the distance to become larger. The same can be observed for a different set of curves in Figure 4.

The integration method used is crucial for the accuracy of the algorithm. The expression in Equation 1 is strongly peaked near the mean locations of the input points, and the function must be sampled sufficiently densely near these locations. Our implementation uses the R package `cubature`[1].

The running time of the algorithm depends highly on the input trajectories and ε. The reason for this is that the heuristic described at the end of Section 4 was implemented, which in some cases terminates the predecessor search much earlier than in the worst case. In general the running time is lower for small ε than for large values. For the examples shown, with $n = 9$, the running time can be up to 1 minute.

[1] https://cran.r-project.org/web/packages/cubature/

6 Conclusion

We discussed the problem of computing the discrete Fréchet distance between polygonal curves with uncertain points. Many interesting questions remain open. The main question is whether the distribution function $F(\varepsilon)$ can be computed efficiently. Another direction for future work is improving the running time of the algorithm presented, either in general or for specific classes of trajectories.

References

[1] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014.

[2] H.-K. Ahn, C. Knauer, M. Scherfenberg, L. Schlipf, and A. Vigneron. Computing the discrete Fréchet distance with imprecise input. *International Journal of Computational Geometry & Applications*, 22(01):27–44, 2012.

[3] R. B. Avraham, O. Filtser, H. Kaplan, M. J. Katz, and M. Sharir. The discrete Fréchet distance with shortcuts via approximate distance counting and selection. In *Proceedings of the thirtieth annual symposium on Computational geometry*, page 377. ACM, 2014.

[4] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 318–337. SIAM, 2012.

[5] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report, Technische Universität Wien, 1994.

[6] C. Fan and B. Zhu. Complexity and algorithms for the discrete Fréchet distance upper bound with imprecise input. *arXiv preprint arXiv:1509.02576*, 2015.

[7] A. Jorgensen, M. Löffler, and J. M. Phillips. Geometric computations on indecisive and uncertain points. *arXiv preprint arXiv:1205.0273*, 2012.

[8] W. H. Press, S. A. Teukolsky, T. Veterling, and B. Flannery. *Numerical recipes: The art of scientific computing*, pages 196–200. Cambridge university press, 3rd edition, 2007.

Mapping polygons to the grid with small Hausdorff and Fréchet distance*

Quirijn W. Bouts† Irina Kostitsyna† Marc van Kreveld§
Wouter Meulemans‡ Willem Sonke† Kevin Verbeek†

1 Introduction

Transforming the representation of objects from the real plane onto a grid has been studied for decades due to its applications in computer graphics, computer vision, and finite-precision computational geometry [8]. Two interpretations of the grid are possible: (i) the grid graph, consisting of vertices at all points with integer coordinates, and horizontal and vertical edges between vertices at unit distance; (ii) the pixel grid, where the only elements are pixels, which are unit squares. In the latter interpretation, one can choose between 4-neighbor or 8-neighbor grid topology.

The issues involved when moving from the real plane to a grid already start with the definition of a line segment on a (pixel) grid, also called a *digital straight segment* [10]. For example, it is already difficult to represent line segments such that the intersection between any pair is a connected set (or empty). More generally, the challenge is to represent objects on a grid in such a way that certain properties of those objects in the real plane transfer to related properties on the grid; connectedness of the intersection of two line segments is an example of this.

While most of the research related to *digital geometry* is done from the graphics or vision perspective, computational geometry has made a number of contributions as well. Besides finite-precision computational geometry [8] these include snap rounding [6, 7, 9], consistent digital rays with small Hausdorff distance [5], mosaic maps [4], and schematization by map matching [11].

We consider the problem of representing a simple polygon P as a polygon in the grid with small distance between them. A *grid cycle* is a simple cycle of edges and vertices of the grid graph corresponding to the grid. A *grid polygon* is a set of pixels whose boundary is a grid cycle. Two of the standard ways of measuring the distance are the Hausdorff distance [1] and the Fréchet distance [2]; we will consider both.

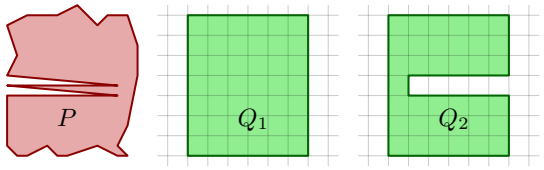


Figure 1: $d_H(P, Q_1)$ is small but $d_H(\partial P, \partial Q_1)$ is not. $d_H(P, Q_2)$ and $d_H(\partial P, \partial Q_2)$ are both small but the Fréchet distance $d_F(\partial P, \partial Q_2)$ is not.

Let X and Y be two subsets of a metric space. The (directed) *Hausdorff distance* $d_H(X, Y)$ from X to Y is defined as the maximum distance from any point in X to its closest point in Y. In Section 2 we show that for any simple polygon P, a grid polygon Q exists with $d_H(P, Q) \leq \frac{1}{2}\sqrt{2}$ and $d_H(Q, P) \leq \frac{3}{2}\sqrt{2}$ on the unit grid. Furthermore, the constructed polygon satisfies the same bounds for the distance between the boundaries ∂P and ∂Q. This is not equivalent, since the point that realizes the maximum smallest distance to the other polygon may lie in the interior (Fig. 1).

Under the Hausdorff distance, the polygon boundary ∂Q does not necessarily intuitively resemble ∂P (Fig. 1, P and Q_2). Therefore, the Fréchet distance d_F [2] between the boundaries may be a better measure for similarity. Unlike the Hausdorff distance, however, not every polygon boundary ∂P can be represented by a grid cycle with constant Fréchet distance. In Section 3 we present a condition on the input polygon boundary related to fatness (in fact, to κ-straightness [3]) and show that it allows a grid cycle representation with constant Fréchet distance.

2 Hausdorff distance

In this section, we present an algorithm that achieves a low Hausdorff distance between both the boundaries and the interiors of the input polygon P and the resulting grid polygon Q. We say that two cells are *adjacent* if they share a segment. If two cells share only a point, then they are *point-adjacent*.

*Research on the topic of this paper was initiated at the 1st Workshop on Applied Geometric Algorithms (AGA 2015) in Langbroek, The Netherlands, supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208.

†Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands, [q.w.bouts|i.kostitsyna|w.m.sonke|k.a.b.verbeek]@tue.nl. Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208 and 639.021.541.

‡giCentre, City University London, United Kingdom, wouter.meulemans@city.ac.uk. Supported by Marie Skłodowska-Curie Action MSCA-H2020-IF-2014 656741.

§Dept. of Information and Computing Sciences, Utrecht University, m.j.vankreveld@uu.nl.

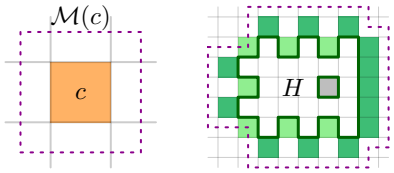


Figure 2: (a) Module $\mathcal{M}(c)$ (dashed) of cell c. (b) Illustration to Lemma 1. $Q_1 \cap B$ in green; $Q_2 \cap B$ in gray; curve C dashed.

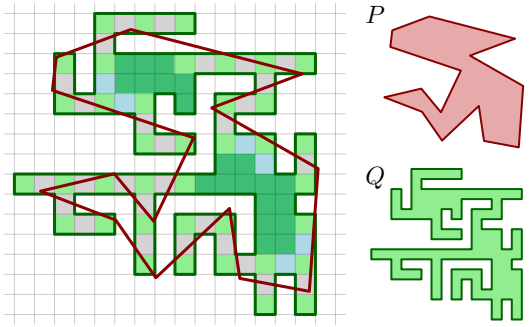


Figure 3: Example of the Hausdorff algorithm; the input and output are shown on the right. Colors: Q_1, Q_2, Q_3, Q_4.

Algorithm. We represent the grid polygon Q as a set of cells (or pixels). If two cells $c_1 \in Q$ and $c_2 \in Q$ are point-adjacent, and there is no cell $c \in Q$ that is adjacent to both c_1 and c_2, then c_1 and c_2 share a *point-contact*. We construct Q as the union of four sets Q_1, Q_2, Q_3, Q_4 (not necessarily disjoint). To define these sets, we define the *module* $\mathcal{M}(c)$ of a cell c as the two-by-two square region centered at the center of c (see Fig. 2(a)). Furthermore, since we can number the rows and columns, we can speak of even-even cells, odd-odd cells, odd-even cells, and even-odd cells. The four sets are defined as follows; see also Fig. 3.

Q_1: All cells c for which $\mathcal{M}(c) \subseteq P$.

Q_2: All even-even cells c for which $\mathcal{M}(c) \cap P \neq \emptyset$.

Q_3: For all cells $c_1, c_2 \in Q_1 \cup Q_2$ that share a point-contact, the two cells that are adjacent to both c_1 and c_2 are in Q_3.

Q_4: A maximal set of cells that does not introduce holes, and where each cell $c \in Q_4$ is adjacent to two cells in Q_2 and $\mathcal{M}(c) \cap P \neq \emptyset$.

We note that the set $Q_1 \cup Q_2$ is sufficient to achieve the desired Hausdorff distance. We add the set Q_3 to resolve point-contacts, and the set Q_4 to make the set Q connected.

Lemma 1 *The set $Q_1 \cup Q_2$ is hole-free, even when including point-adjacencies.*

Proof. For the sake of contradiction, let H be a maximal set of cells comprising a hole. Consider the set B of all cells in $Q_1 \cup Q_2$ that surround H and are adjacent to a cell of H. Since Q_2 contains only even-even cells, every cell in $Q_2 \cap B$ must be (point-)adjacent to two cells in $Q_1 \cap B$ (see Fig. 2(b)). Hence, the boundary of the union of all modules of cells in $Q_1 \cap B$ is a single closed curve C; if this union contains a hole, P would contain a hole as well. Since $C \subset P$ due to the definition of Q_1, the interior of C must also be in P. Finally note that C is a rectilinear curve through the centers of cells, but not through the center of a cell in H. Hence, the module of a cell in H is completely inside C, implying $H \subset Q_1$; this contradicts our assumption. □

Lemma 2 *The set Q is simply connected and does not contain point-contacts.*

Proof. Consider a point-contact between two cells $c_1, c_2 \in Q_1 \cup Q_2$ and a cell $c \notin Q_1 \cup Q_2$ that is adjacent to both c_1 and c_2 ($c \in Q_3$). Since Q_2 contains only even-even cells, we may assume that $c_1 \in Q_1$. Recall that $\mathcal{M}(c_1) \subseteq P$ by definition. We may further assume that c_1 is an odd-odd cell, for otherwise a cell in Q_2 would eliminate the point-contact. Hence, all cells point-adjacent to c_1 are in $Q_1 \cup Q_2$, and thus c has three adjacent cells in $Q_1 \cup Q_2$. This implies that adding $c \in Q_3$ to $Q_1 \cup Q_2$ cannot introduce point-contacts or holes. Similarly, cells in Q_4 connect two oppositely adjacent cells in Q_2, and thus cannot introduce point-contacts (or holes, by definition). Combining this with Lemma 1 implies that Q is hole-free and does not contain point-contacts.

It remains to show that Q is connected. For the sake of contradiction, assume that Q is not connected, so take two cells c_1 and c_2 in different connected components. We may further assume that $c_1, c_2 \in Q_2$, as cells in $Q_1 \cup Q_3 \cup Q_4$ must be adjacent or point-adjacent to a cell in Q_2. Let $p \in \mathcal{M}(c_1) \cap P$, $q \in \mathcal{M}(c_2) \cap P$ and consider a path π between p and q inside P. Every even-even cell c with $\mathcal{M}(c) \cap \pi \neq \emptyset$ must be in Q_2. Furthermore, the modules of even-even cells cover the plane. Thus, there must be two cells $c, c' \in Q_2$ in different components such that the module of the cell adjacent to both c and c' intersects π. This contradicts the maximality of Q_4. □

Upper bounds. To prove our bounds, note that $\mathcal{M}(c) \cap P \neq \emptyset$ holds for every cell $c \in Q$. This is explicit for cells in Q_1, Q_2, and Q_4. For cells in Q_3, note that these cells must be adjacent to a cell in Q_1, and thus contain a point in P.

Lemma 3 $d_H(P, Q), d_H(\partial P, \partial Q) \leq \frac{1}{2}\sqrt{2}$.

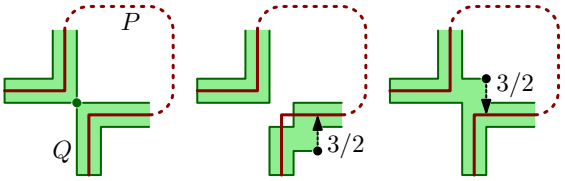


Figure 4: A polygon that does not admit a grid polygon with Hausdorff distance smaller than 3/2. The brown line signifies a very thin polygon.

Proof. Let $p \in P$ and consider the even-even cell c such that $p \in \mathcal{M}(c)$. Since $c \in Q_2$, the distance $d_H(p, Q) \leq d_H(p, c) \leq \frac{1}{2}\sqrt{2}$. Now consider a point $p \in \partial P$. There is a 2×2-set of cells whose modules contain p. This set contains an even-even cell $c \in Q$ and an odd-odd cell $c' \notin Q$. The latter is true, because odd-odd cells in Q must be in Q_1. Therefore, the point q shared by c and c' must be in ∂Q. Thus, $d_H(p, \partial Q) \leq d_H(p, q) \leq \frac{1}{2}\sqrt{2}$. □

Lemma 4 $d_H(Q, P), d_H(\partial Q, \partial P) \leq \frac{3}{2}\sqrt{2}$.

Proof. Let $q \in Q$ and let $c \in Q$ be the cell that contains q. Since $\mathcal{M}(c) \cap P \neq \emptyset$, we can choose a point $p \in \mathcal{M}(c) \cap P$. It directly follows that $d_H(q, P) \leq d_H(q, p) \leq \frac{3}{2}\sqrt{2}$. Now consider a point $q \in \partial Q$, and let $c \in Q$ and $c' \notin Q$ be two adjacent cells such that $q \in \partial c \cap \partial c'$. We claim that $(\mathcal{M}(c) \cup \mathcal{M}(c')) \cap \partial P \neq \emptyset$. If $c \notin Q_1$, then the claim directly follows. Otherwise, $\mathcal{M}(c) \subseteq P$ implies that $\mathcal{M}(c') \cap P \neq \emptyset$ and clearly $\mathcal{M}(c') \nsubseteq P$. This in turn implies that $\mathcal{M}(c') \cap \partial P \neq \emptyset$. Let $p \in (\mathcal{M}(c) \cup \mathcal{M}(c')) \cap \partial P$. Then $d_H(q, \partial P) \leq d_H(q, p) \leq \frac{3}{2}\sqrt{2}$. □

Theorem 5 *For every simple polygon P there exists a simply connected grid polygon Q without point-contacts such that $d_H(P, Q), d_H(\partial P, \partial Q) \leq \frac{1}{2}\sqrt{2}$ and $d_H(Q, P), d_H(\partial Q, \partial P) \leq \frac{3}{2}\sqrt{2}$.*

Lower bounds. In Fig. 4 a polygon is shown for which no grid polygon has Hausdorff distance below 3/2 between the boundaries or interiors. A naive construction of a grid polygon results in the left drawing of Fig. 4 which is not a simple polygon. To make it simple, we can either remove a cell (center) or add a cell (right). Both methods result in $d_H(\partial Q, \partial P) \geq 3/2 - \epsilon$, for any $\epsilon > 0$. Alternatively, we can fill the entire upper-right part of the grid polygon (not shown), resulting in a high $d_H(Q, P)$.

In the L_∞ distance, the lower bound given in Fig. 4 also holds. Interestingly, in this measure, our algorithm achieves a Hausdorff distance of 3/2 (the upper-bound proofs can be straightforwardly modified to show this).

3 Fréchet distance

The Fréchet distance d_F between two curves is generally a better measure for similarity than the Hausdorff distance; see [2] for a definition of the measure. We consider computing a grid polygon Q whose boundary has constant Fréchet distance to the boundary of the input polygon P. We study under what conditions on ∂P this is possible and prove a bound.

Obesity. Some input polygons P do not admit a grid polygon Q such that their boundaries have low Fréchet distance; see for example the polygon in Fig. 7(a). Intuitively, any grid polygon boundary ∂Q approximating ∂P must significantly deviate from it, because the grid is too coarse to follow ∂P closely.

However, this problem is caused only by the thin spikes: if we assume that P does not have those, we can do better. Let $|ab|_{\partial P}$ be the distance from a to b along ∂P. As defined in [3], a curve C is *κ-straight* if for any two points $a, b \in C$, $|ab|_C \leq \kappa \cdot |ab|$. In fact, we need this property on ∂P only when $|ab| \leq \sqrt{2}$, as we must deal with several parts of ∂P being in the same grid cell. We therefore define a weaker fatness measure called *β-obesity*: a polygon P is *β-obese* if for any two points $a, b \in \partial P$ with $|ab| \leq \sqrt{2}$, $|ab|_{\partial P} \leq \beta$.

Algorithm. The algorithm constructs Q via a grid cycle C representing ∂Q. Consider for all grid graph vertices a 1×1-square centered on the vertex, and let C' be the cyclic chain of vertices whose square is intersected by ∂P, in the order in which ∂P visits them (see Fig. 5). Note that C' may contain duplicates. Now C is obtained by iteratively finding a duplicate with minimal distance along the curve between the two occurrences, and removing the corresponding subchain. After all the duplicates are removed, the grid polygon Q with boundary C is returned, unless C encloses no cells. In that case C is a single vertex

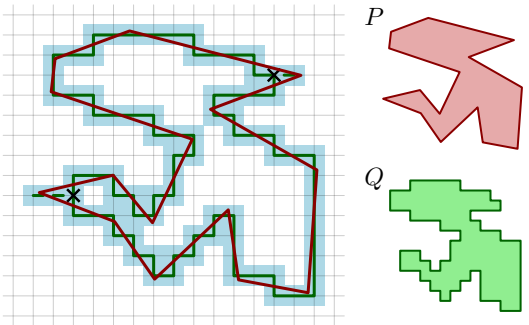


Figure 5: Example of the Fréchet algorithm; the input and output are shown on the right. The two crosses mark points appearing in C twice, hence their subpaths (shown dashed) are removed.

or two vertices connected by one edge, and we let the grid polygon Q consist of a single cell intersecting P.

Upper bounds. Q cannot contain duplicate vertices, so it must be a grid polygon. Therefore we need to prove only the bound on the Fréchet distance.

Theorem 6 *Given a β-obese polygon P with $\beta \geq \sqrt{2}$, there exists a grid polygon Q such that $d_F(\partial P, \partial Q) \leq (\beta + \sqrt{2})/2$.*

Proof. Consider C (representing ∂Q) as obtained by the algorithm described above (ignoring the case where C did not enclose cells). We will show that $d_F(\partial P, C) \leq (\beta + \sqrt{2})/2$. We will define a mapping between ∂P and C that gives rise to the parameterizations of ∂P and C, needed to bound the Fréchet distance, and show that the distance between mapped points is at most $(\beta + \sqrt{2})/2$.

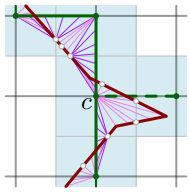


Figure 6: Mapping between C and ∂P.

First we define a mapping between ∂P and the vertices of C' in the natural way: by proximity. We map the edges of C' to points of ∂P, namely to the points where ∂P intersects a boundary of a 1×1-square centered on a vertex of C'. This mapping is also simply by proximity. We convert this mapping into one between ∂P and C: Whenever we remove a subchain from C', that whole subchain is mapped to the vertex that is the start and end of that subchain (refer to Fig. 6). Once C is obtained, only a single connected component of ∂P is mapped to any vertex of C and only one edge of C is mapped to any point of ∂P. The resulting mapping is monotone by construction.

Consider any vertex c of C. If ∂P visits the 1×1-square s of c only once, then exactly the part of ∂P inside s maps to c, and the distance between c and the part of ∂P mapped to it is at most $\sqrt{2}/2$. If ∂P visits s twice, then the part of ∂P outside s between these visits is also mapped to c. The length of this boundary external to s is at most β, so its furthest point is at most $\beta/2$ away from s and hence at most $\beta/2 + \sqrt{2}/2$ from c, leading to the desired bound. When ∂P visits s more than twice, the same argument can be used.

Finally, the distance between edges of C and points of ∂P is at most $\sqrt{2}/2$, which is easy to see. □

Lower bound. Though we omit a full proof of our lower bound, its essence lies with constructing a polygon as sketched in Fig. 7(a). The border ∂Q of a grid polygon with low Fréchet distance to ∂P needs to follow the spikes in ∂P. However, as the grid is too coarse, there is not enough vertical space to do so (Fig. 7(b)). By using spikes of length linear in β, we get the bound claimed below in Theorem 7.

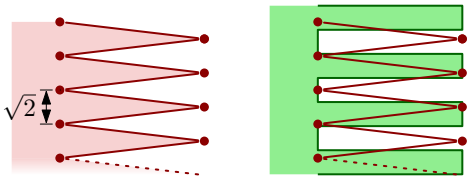


Figure 7: A polygon (left) for which any grid polygon will have high Fréchet distance (right).

Theorem 7 *For any $\beta > \sqrt{2}$, there exists a β-obese polygon P for which for any grid polygon Q, $d_F(\partial P, \partial Q) \geq \frac{1}{4}\sqrt{\beta^2 - 2}$.*

References

[1] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell.*, 13(3-4):251–265, 1995.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. & Appl.*, 5:75–91, 1995.

[3] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[4] R. G. Cano, K. Buchin, T. Castermans, A. Pieterse, W. Sonke, and B. Speckmann. Mosaic drawings and cartograms. *Comp. Graph. Forum*, 34(3):361–370, 2015.

[5] J. Chun, M. Korman, M. Nöllenburg, and T. Tokuyama. Consistent digital rays. *Discr. & Comput. Geom.*, 42(3):359–378, 2009.

[6] M. de Berg, D. Halperin, and M. H. Overmars. An intersection-sensitive algorithm for snap rounding. *Comput. Geom.*, 36(3):159–165, 2007.

[7] M. T. Goodrich, L. J. Guibas, J. Hershberger, and P. J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th SoCG*, pages 284–293, 1997.

[8] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *Proc. 27th FOCS*, pages 143–152, 1986.

[9] J. Hershberger. Stable snap rounding. *Comput. Geom.*, 46(4):403–416, 2013.

[10] R. Klette and A. Rosenfeld. Digital straightness – a review. *Discr. Appl. Math.*, 139(1-3):197–230, 2004.

[11] W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization.* PhD thesis, Technische Universiteit Eindhoven, 2014.

Map Simplification with Topology Constraints: Exactly and in Practice

Stefan Funke Thomas Mendel Alexander Miller Sabine Storandt Maria Wiebe

Abstract

We consider the classical line simplification problem subject to a given error bound ϵ but with additional topology constraints as they arise for example in the map rendering domain. While theoretically inapproximability has been proven for these problem variants, we show that in practice one can solve medium sized instances optimally using an integer linear programming approach and larger instances using an heuristic approach which for medium-sized real-world instances yields close-to-optimal results. Our approaches are evaluated on data sets which are synthetically generated, stem from the OpenStreetMap project, and the 2014 GIS Cup competition.

1 Introduction

In the classical line simplification problem (**CLSP**) we are given a polygonal chain $C = p_0p_1p_2 \dots p_n$ with $p_i \in \mathbb{R}^2$, an error parameter $\epsilon > 0$ and ask for a simplification of C, that is, indices $i_1 < i_2 < \dots < i_k$ with $0 < i_j < n$ such that the polygonal chain $\widetilde{C} = p_0p_{i_1}p_{i_2} \dots p_{i_k}p_n$ is a faithful approximation of C. Here 'faithful' means that for every 'shortcut' segment $s_j = p_{i_j}p_{i_{j+1}}$ of the simplification the furthest distance of a point in $\{p_{i_j+1}, \dots p_{i_{j+1}-1}\}$ to the shortcut segment s_j is at most ϵ. A natural optimization goal is to compute a faithful approximation with as few vertices as possible, that is, minimizing k.

Solving CLSP is of great interest in particular in the map rendering context. One of the main challenges for rendering map data on the screen arises from the abundance of data. Assume we want to render the road network of Germany on a mobile device like a tablet. A cross-country Autobahn like the A7 consists of several thousands of individual road segments. Rendering all of them is certainly a waste of time when dealing with the screen of a mobile device. So typically one would *simplify* the chain of segments by replacing subsequences of degree-2 nodes along the A7 by single road segments. Depending on the screen size and resolution, this can be done without really affecting the visual quality of the result. Naturally, this simplification should not introduce self-intersections, so a sensible generalization of CLSP to the map rendering context (originally when simplifying country boundaries) is the map simplification problem (**MSP**), where we are given a planar subdivision in form of a planar embedding of a straight-line graph $G(V, E)$, a parameter $\epsilon > 0$ and the goal is to solve CLSP for each degree-2 chain of the graph such that the total number of surviving vertices is minimized without introducing intersections (within a single degree-2 chain as well as between different degree-2 chains).

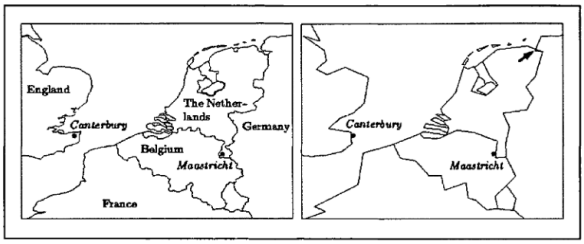


Figure 1: A map of Western Europe with inconsistencies after line simplification of country boundaries (from [1]), courtesy of de Berg et al.

Unfortunately, just solving MSP without additional care might lead to undesired effects, see Figure 1. In this simplification (right) of a map excerpt of Europe (left), some cities switched countries or ended up in the sea. This gives rise to a more general map simplification problem with topology constraints (**MSTOPOP**): Given a planar subdivision as a planar embedding of a straight-line graph $G(V, E)$, a parameter $\epsilon > 0$ and a set of points $P \subset \mathbb{R}^2$, the goal is to solve CLSP for each maximal degree-2 chain of the graph such that the total number of surviving vertices is minimized, no intersections are introduced, and every point $p \in P$ remains in the same face as before.

For our map rendering application, MSTOPOP is the most natural formulation of the respective optimization problem. Nevertheless, to allow for simpler solution strategies and efficient solution we will define a more local variant of this problem called **MSLOC-TOPOP** in Section 2 (which still turns out to be theoretically hard to solve and even approximate).

1.1 Related Work

For CLSP there are several known algorithms, the most popular being the algorithm by Douglas/Peucker [2], which unfortunately does not guarantee absence of self-intersections nor optimality (i.e.

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

minimum number of surviving points) of the result. Its worst-case running time is $\Theta(n^2)$, though better running times are experienced in practice. The algorithm by Imai/Iri [5] guarantees as a result a minimum number of surviving points, but not absence of self-intersections. Its running time is $O(n^3)$ in its original version, but faster variants with $O(n^2)$ running time exist. Estkowski and Mitchell [4] have shown, that from a theoretical point of view, solving MSP or MSTOPOP optimally is a hopeless enterprise. They prove that for MSTOPOP it is NP-hard to obtain an approximate solution better than within a factor of $n^{1/5-\delta}$ for any $\delta > 0$. Their result carries over to MSP since topology constraint points do not play a role in their proof of approximation-hardness. In [1] de Berg et al. consider heuristic solutions to MSTOPOP and MSP, yet without a comparison with the respective optimum solutions. For MSP, an implementation is available in the CGAL library [6] following [3]. To our knowledge, no study has been conducted investigating how close to the optimum heuristic solutions are for MSTOPOP (due to lack of an exact solution).

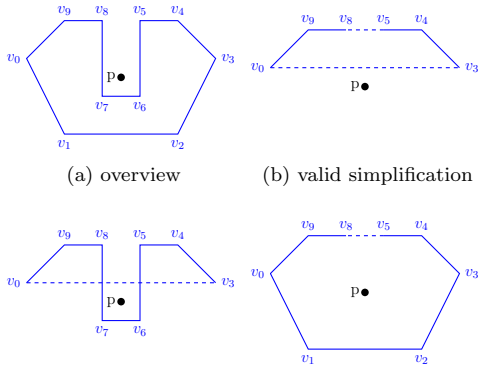
As a side note, during the GISCup'14 – a competition held during the ACM SIGSPATIAL conference 2014 – a variant of the problem (without a precision constraint – i.e., $\epsilon = \infty$) was tackled by several teams.

1.2 Our Contribution

We define a *local* variant of the map simplification with topology constraints problem (which theoretically is still hard to approximate) and derive a respective ILP formulation which can solve instances of moderate size optimally. We then develop a heuristic algorithm based on constrained triangulations and local simplification steps which empirically can be shown to produce close-to-optimal results for moderately sized instances (via comparison to the ILP solution). In contrast to the ILP solution this heuristic can also be used to solve large instances as they naturally occur in the map rendering domain.

2 Local Topology-Consistency

At first sight one might think that solving MSTOPOP is exactly what we want for our map rendering application. Consider the example in Figure 2(a), where we have a planar subdivision with two faces – one U-shaped face bounded by $v_0v_1 \dots v_9v_0$ and an outer face which also contains a topology constraint point p. For sufficiently large value of ϵ, the simplification shown in 2(b) is indeed a valid simplification according to MSTOPOP since p still lies in the outer face. This might be somewhat counterintuitive since p somehow 'switched sides' (even though topologically it is, of course, still on the right side). In particular, if we locally inspect the shortcut v_0v_3 which replaces the



(a) overview (b) valid simplification

(c) self-intersection (d) topology constraint violated

Figure 2: Example of two simplifications, that are not valid alone but only in conjunction with each other.

chain $v_0v_1v_2v_3$ there is indeed a switch of sides (c) – which is only healed topologically by shortcutting $v_5v_6v_7v_8$ by v_5v_8, which also is invalid on its own (d).

We believe that it is not unnatural to demand that shortcuts *locally* don't make points switch sides. To that end we define the following local criterion to decide whether a shortcut is considered topology preserving.

Definition 1 *For given $\epsilon > 0$ and constraint point set P, a shortcut u_1u_k is considered a* valid shortcut *for the polygonal chain $C = u_1u_2 \dots u_k$ if*

- *the distance of u_i, $1 < i < k$ to the segment u_1u_k is at most ϵ.*
- *the polygon (possibly with self-intersections) defined by the polygonal chain $C' = u_1u_2 \dots u_ku_1$ does not contain[1] a constraint point.*

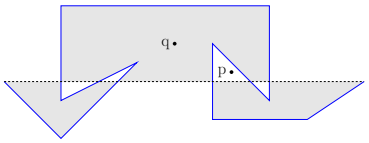


Figure 3: Example of a possible simplifciation. Point q would make this simplification locally inconsistent, whereas point p would allow this simplification. The interior of the polygon is given by the shaded area.

See Figure 3 for an illustration of this definition. Armed with this notion of a valid shortcut we can formally define the map simplification variant that we will be dealing with in the following.

Definition 2 (MSLOCTOPOP) *For a planar subdivision given as a straight-line embedding of a graph*

[1] With the interior of a (possibly complex) polygon defined by the *even-odd rule*. See Figure 3 for an example.

$G(V, E)$, a set of constraint points $P \subset \mathbb{R}^2$, and an $\epsilon > 0$, the goal of the Map Simplification with LOCal TOPOlogy constraints Problem (MSLOCTOPOP) *is to simplify degree-2 chains of G using non-intersecting valid shortcuts such that the total number of remaining vertices is minimized.*

As MSLOCTOPOP comprises MSP as a special case (no topology constraints), the hardness of approximation result in [4] carries over, hence there is little hope to find a polynomial-time approximation algorithm which solves MSLOCTOPOP with an approximation ratio substantially better than $n^{1/5}$.

3 An Integer Linear Programming Formulation for MSLOCTOPOP

As we have seen, even our specialization MSLOCTOPOP of MSTOPOP is hard to approximate, yet using an integer linear programming (ILP) formulation one might be able to obtain optimal solutions for many instances that occur in practice. We will develop a respective ILP in the following step by step.

Let us first concentrate on a single polygonal chain $C^l = p_1 p_2 \dots p_{n_l}$ of the planar subdivision where each p_i with $1 < i < n_l$ is a degree-2 node of the subdivision, p_1 and p_{n_l} are nodes with degree $\neq 2$. We first construct the set $S^l := \{s_1^l, s_2^l, \dots s_{k_l}^l\}$ of *valid shortcuts* for C^l. Note that the original edges are also valid shortcuts and $k_l \in O(n_l^2)$. Essentially we want to construct a path from p_1 to p_{n_l} using as few valid shortcuts as possible, so we introduce 0-1 variables $x_1^l, x_2^l, \dots x_{k_l}^l$ where $x_i^l = 1$ denotes that the shortcut s_i^l should be realized, $x_i^l = 0$ that it should not be used. As constraints we demand:

$$\sum_{s_j^l = (p_1, .)} x_j^l = 1 \tag{1}$$

$$\sum_{s_j^l = (., p_{n_l})} x_j^l = 1 \tag{2}$$

that is, we select exactly one shortcut from S^l that is adjacent to p_1 and likewise for p_{n_l}. For every other vertex p_i, $1 < i < n_l$ of C^l we want that the number of incoming shortcuts equals the number of outgoing shortcuts (in fact both equal to 0 or to 1, but in our case there is no need to explicitly enforce that):

$$\forall 1 < i < n_l : \sum_{s_j^l = (., p_i)} x_j^l - \sum_{s_j^l = (p_i, .)} x_j^l = 0 \tag{3}$$

We construct variables and respective constraints for each polygonal degree-2 chain in the planar subdivision. Then for every *intersecting* pair of shortcuts s_i^l, s_j^g (of the same or different polygonal degree-2 chains) we add a constraint

$$x_i^l + x_j^g \leq 1 \tag{4}$$

preventing the usage of both shortcuts simultaneously. The objective function is simply a minimization of the sum of all variables

$$\min \sum x_i^l \tag{5}$$

MSLOCTOPOP being NP-hard to approximate, we cannot expect our ILP formulation to be solvable efficiently for every input instance. Yet, instances occurring in real-world scenarios might well be solvable with a good ILP solver.

4 A Local Simplification Heuristic

In this section we present a heuristic to solve the MSLOCTOPOP problem. We iteratively remove single points of the subdivision by only inspecting local neighborhoods, yet preserving validity of the overall simplification. The basic idea is similar to [3] but also incorporates topology constraints.

We employ a *Constrained Triangulation (CT)* with the points being all vertices of the original subdivision as well as the constraint points, and the constraining edges being the edges of the subdivision. Let $\deg_G(v)$ denote the number the constraining edges adjacent to a node in the current CT.

Now for a given node v with $\deg_G(v) = 2$ one can quickly decide whether it can be removed and replaced by a shortcut without violating any topology constraint or creating intersections. Let v_1 and v_2 be the two neighbours of v in the current subdivision. There are two cases for which we can easily see, that we may *not* discard v:

- The nodes v, v_1 and v_2 form a triangle in the current planar subdivision. Removing v leads to a collapse of the 2-dimensional face spanned by the 3 nodes.
- The distance of v to the segment $\overline{v_1 v_2}$ is greater than ϵ. Then the shortcut $v_1 v_2$ is not valid.

If none of these cases applies we also demand that there exists no point $u \neq v_1, v_2$ adjacent (in the triangulation) to v which lies in the triangle $\triangle v_1 v v_2$, otherwise:

- If u is part of the constraint points in the subdivision, the simplification is invalid according to our *local topology-consistency*.
- If u is part of the subdivision boundary, the removal of v either introduces an intersection or changes the orientation of a face.

Given these criteria the complete algorithm is quite simple. For every point we compute whether it is removable with respect to these criteria and remove it if this is the case. This procedure is repeated until no more points can be removed.

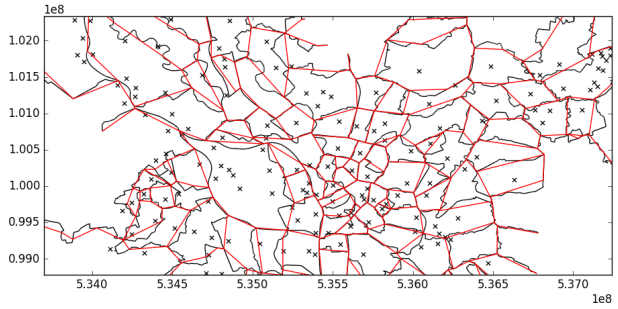


Figure 4: Hamburg data set, $\epsilon = 10^5$.

Note that we have to make sure, that the distance check is performed with respect to *all* the points that have possibly been replaced by a shortcut.

5 Experimental Results

We have compared both solution approaches on synthetic and real-world data, yet due to space restrictions we only report on some selected instances. The experiments were run on a standard laptop with an Intel Core i5/1.9GHz/12GB RAM. The local simplification heuristic uses the CGAL library [6], the ILPs were solved using the Gurobi solver. Clang 3.7 with the -O3 flag was used for compilation. From the OpenStreetMap project we extracted the datasets **HAM** – the administrative subdivision of the city of Hamburg with some POIs as topology constraint points, see Figure 4 – and **GMNY** – the country borders for Germany with all cities and towns as constraint points. From the GISCup'14 (`http://mypages.iit.edu/~xzhang22/GISCUP2014/`), the planar subdivisions **GIS4** and **GIS5** including topology constraint points were used.

Table 1 lists the results. For example, for a tolerance of $\epsilon = 10^4$ and the dataset HAM, the ILP approach reduces the number of surviving degree-2-nodes to 992 within 19 seconds. The heuristic approach, on the other hand takes only 0.4 seconds to obtain a result with 1219 surviving degree-2-nodes. So while not as small as the optimum ILP result, the heuristic result is reasonably close. For the large GMNY instance, the ILP approach could not determine a solution within one hour whereas the heuristic produced a solution within 14 seconds. For GIS4 and GIS5, the heuristic also produces solutions pretty close to the ILP optimum. For the latter two data sets we also had running times and result sizes of the runner-up algorithm at the 2014 GISCup – here called CROSS (we could not get hands on the winning algorithm). Note though, that the objective of the GISCup was not just minimization of the remaining subdivision but rather the ratio of removed points per unit of time. So while being blazingly fast, CROSS retains a lot more points even compared to our heuristic approach.

	GIS4	GIS5	HAM	GMNY
# nodes	26198	25203	10233	217863
# constraints	356	1607	194	97639
$\epsilon = 10000$				
ILP time (s)	296	105	19	-
Heur. time(s)	1.3	1.1	0.4	14
ILP output	348	476	992	-
Heur. output	433	566	1219	≈11k
$\epsilon = 100000$				
ILP time (s)	419	121	56	-
Heur. time(s)	1.3	1.2	0.4	17
ILP output	88	238	57	-
Heur. output	101	275	67	1864
$\epsilon = \infty$				
ILP time (s)	439	130		
Heur. time(s)	1.4	1.2		
CROSS time(s)	0.01	0.01		
ILP output	88	237		
Heur. output	100	274		
CROSS output	1759	2826		

Table 1: Pruning results for our algorithms: running times and size of the output.

References

[1] Marc de Berg, Marc van Kreveld, and Stefan Schirra. A new approach to subdivision simplification. In *ACSM/ASPRS Annual Convention & Exposition Technical Papers*, pages 79–88, Charlotte, North Carolina, USA, 1995. ACSM.

[2] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[3] Christopher Dyken, Morten Dæhlen, and Thomas Sevaldrud. Simultaneous curve simplification. *J. of Geographical Systems*, 11(3):273–289, 2009.

[4] Regina Estkowski and Joseph S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proc. 17th Ann. Symp. on Comp. Geom.*, SCG '01, pages 40–49. ACM, 2001.

[5] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. Elsevier Science, 1988.

[6] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.7 edition, 2015.

Ramsey-type theorems for lines in 3-space

Jean Cardinal* Michael S. Payne† Noam Solomon‡

Abstract

We prove geometric Ramsey-type statements on collections of lines in 3-space. These statements give guarantees on the size of a clique or an independent set in (hyper)graphs induced by incidence relations between lines, points, and reguli in 3-space. Among other things, we prove the following:

- The intersection graph of n lines in $\mathbb{R}^3$ has a clique or independent set of size $\Omega(n^{1/3})$.
- Every set of n lines in $\mathbb{R}^3$ has a subset of $\sqrt{n}$ lines that are all stabbed by one line, or a subset of $\Omega\left((n/\log n)^{1/5}\right)$ lines such that no 6-subset is stabbed by one line.
- Every set of n lines in general position in $\mathbb{R}^3$ has a subset of $\Omega(n^{2/3})$ lines that all lie on a regulus, or a subset of $\Omega(n^{1/3})$ lines such that no 4-subset is contained in a regulus.

The proofs of these statements all follow from geometric incidence bounds – such as the Guth-Katz bound on point-line incidences in $\mathbb{R}^3$ – combined with Turán-type results on independent sets in sparse graphs and hypergraphs. As an intermediate step towards the third result, we also show that for a fixed family of plane algebraic curves with s degrees of freedom, every set of n points in the plane has a subset of $\Omega(n^{1-1/s})$ points incident to a single curve, or a subset of $\Omega(n^{1/s})$ points such that at most s of them lie on a curve. Although similar Ramsey-type statements can be proved using existing generic algebraic frameworks, the lower bounds we get are much larger than what can be obtained with these methods. The proofs directly yield polynomial-time algorithms for finding subsets of the claimed size.

1 Introduction

Ramsey theory studies the conditions under which particular discrete structures must contain certain substructures. Ramsey's Theorem says that for every n, every sufficiently large graph has either a clique or an independent set of size n. Early geometric Ramsey-type statements include the Happy Ending Problem on convex quadrilaterals in planar point sets, and the Erdős-Szekeres Theorem on subsets in convex position [7].

We prove a number of Ramsey-type statements involving lines in $\mathbb{R}^3$. Our proofs combine two main ingredients: geometric information in the form of bounds on the number of incidences among the objects, and a Turán-type theorem that converts this information into a Ramsey-type statement.

Ramsey's Theorem for graphs and hypergraphs only guarantees the existence of rather small cliques or independent sets. However for the geometric relations we study the bounds are known to be much larger. Therefore we are interested in finding the correct asymptotics. In particular, we are interested in the *Erdős-Hajnal property*. A class of graphs has this property if each member with n vertices has either a clique or an independent set of size n^δ for some constant $\delta > 0$. The results presented here make use of important recent advances in combinatorial geometry, a key example of which is the bound on the number of incidences between points and lines in $\mathbb{R}^3$ given by Guth and Katz [10] in their recent solution of the Erdős distinct distances problem.

1.1 A general framework

In general we consider two classes of geometric objects $\mathcal{P}$ and $\mathcal{Q}$ in $\mathbb{R}^d$ and a binary incidence relation contained in $\mathcal{P} \times \mathcal{Q}$. For a finite set $P \subseteq \mathcal{P}$ and an integer $t \geq 2$, we say that a t-subset $S \in \binom{P}{t}$ is *degenerate* whenever there exists $q \in \mathcal{Q}$ such that every $p \in S$ is incident to q. Hence the incidence relation together with the integer t induces a t-uniform hypergraph $H = (P, E)$, where $E \subseteq \binom{P}{t}$ is the set of all degenerate t-subsets of P. A clique in this hypergraph is a subset $S \subseteq P$ such that $\binom{S}{t} \subseteq E$. Similarly, an independent set is a subset $S \subseteq P$ such that $\binom{S}{t} \cap E = \emptyset$.

In what follows, the families $\mathcal{P}$ and $\mathcal{Q}$ will mostly consist of lines or points in 3-space. We are interested in Erdős-Hajnal properties for the t-uniform hypergraph H.

1.2 Previous results

When $\mathcal{P}$ is a set of points, finding a large independent set amounts to finding a large subset of points in

*Université libre de Bruxelles (ULB), jcardin@ulb.ac.be
†University of Melbourne, michael.payne@unimelb.edu.au
‡Tel Aviv University, noamsolomon@post.tau.ac.il

This is an extended abstract of a presentation given at EuroCG 2016. 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 in a conference with formal proceedings and/or in a journal.

some kind of general position defined with respect to $\mathcal{Q}$. When $\mathcal{Q}$ is the set of points, we are dealing with intersections between the objects in $\mathcal{P}$. In particular, the case $t = 2$ corresponds to the study of geometric intersection graphs.

A set in $\mathbb{R}^d$ is usually said to be in general position whenever no $d + 1$ points lie on a hyperplane. For points and lines in the plane, Payne and Wood proved that the Erdős-Hajnal property essentially holds with exponent 1/2 [16]. Cardinal et al. proved an analogous result in $\mathbb{R}^d$ [3].

Theorem 1 ([16, 3]) *Fix $d \geq 2$. Every set of n points in $\mathbb{R}^d$ contains $\sqrt{n}$ cohyperplanar points or $\Omega((n/\log n)^{1/d})$ points in general position.*

In both cases, the proofs rely on incidence bounds, in particular the Szemerédi-Trotter Theorem [19] in the plane, and the point-hyperplane incidence bounds due to Elekes and Tóth [6] in $\mathbb{R}^d$. We streamlined the technique used in those proofs in order to easily apply it to other incidence relations.

A survey of Erdős-Hajnal properties for geometric intersection graphs was produced by Fox and Pach [8]. A general Ramsey-type statement for the case where $\mathcal{P}$ is the set of plane convex sets was proved by Larman et al. [14] more than 20 years ago. They showed that any family of n such sets contained at least $n^{1/5}$ members that are either pairwise disjoint or pairwise intersecting. Larman et al. also showed that there exist arrangements of $k^{2.3219}$ line segments with at most k pairwise crossing and at most k pairwise disjoint segments. This lower bound was improved successively by Károlyi et al. [12], and Kyncl [13].

More recently Fox and Pach studied intersection graphs of a large variety of other geometric objects [9]. In particular, they proved the Erdős-Hajnal property for families of s-intersecting curves in the plane – families such that no two curves cross more than s times. Erdős-Hajnal properties for hypergraphs have been proved by Conlon, Fox, and Sudakov [5].

A very general version of the problem for the case $t = 2$ has been studied by Alon et al. [1]. Here Ramsey-type results are provided for intersection relations between semialgebraic sets of constant description complexity in $\mathbb{R}^d$. It was shown that intersection graphs of such objects always have the Erdős-Hajnal property. The proof combines a linearisation technique with a space decomposition theorem due to Yao and Yao [20]. As an example, Alon et al. applied their machinery to prove that every family of n pairwise skew lines in $\mathbb{R}^3$ contains at least $k \geq n^{1/6}$ elements $\ell_1, \ell_2, \ldots, \ell_k$ such that ℓ_i passes above ℓ_j for all $i < j$. For the problems we consider, however, the exponents we obtain are significantly larger than what can be obtained from this method.

A more general version of this problem for arbitrary values of t has recently been studied by Conlon et al. [4], for which the Ramsey numbers grow like towers of height $t - 1$.

1.3 Our results

Section 2 deals with the case where $\mathcal{P}$ and $\mathcal{Q}$ are lines and points in $\mathbb{R}^3$. A natural object to consider is the intersection graph of lines in $\mathbb{R}^3$, for which we prove the Erdős-Hajnal property with exponent 1/3. This makes use of the Guth-Katz incidence bound between points and lines in $\mathbb{R}^3$ [11].

Section 3 deals with the setting where both $\mathcal{P}$ and $\mathcal{Q}$ are lines in $\mathbb{R}^3$. We prove that every set of n lines in $\mathbb{R}^3$ has a subset of $\sqrt{n}$ lines that are all stabbed by one line, or a subset of $\Omega\left((n/\log n)^{1/5}\right)$ such that no 6-subset is stabbed by one line. The proof involves the Ramsey-type result on points and hyperplanes due to Cardinal et al. [3], which in turn relies on a point-hyperplane incidence bound due to Elekes and Tóth [6].

Finally, in Section 4 we introduce the notion of a subset of lines in general position in $\mathbb{R}^3$ with respect to reguli, defined as loci of lines intersecting three pairwise skew lines. This uses the Pach-Sharir bound on incidences between points and curves in the plane [15].

The large subsets whose existence our results guarantee can be found in polynomial time.

Omitted proofs are given in a long version of the paper[1].

2 Points and lines in $\mathbb{R}^3$

We consider the setting in which the family $\mathcal{P}$ is the set of lines in $\mathbb{R}^3$ and $\mathcal{Q} = \mathbb{R}^3$. The first subcase we consider is $t = 2$, or in other words, intersection graphs of lines.

Theorem 2 *The intersection graph of n lines in $\mathbb{R}^3$ has a clique or independent set of size $\Omega(n^{1/3})$.*

We now sketch the proof, that combines Turán's Theorem with the Guth-Katz bounds [11, Theorem 4.5] and [11, Theorem 2.11]. The latter can be shown to yield the following.

Lemma 3 *Given a set L of n lines, so that no plane or regulus contains more than s lines, and no point is incident to more than ℓ lines of L, the number of line-line incidences is $O(n^{3/2}\log \ell + ns + n\ell)$.*

(We recall that $\omega(G)$ and $\alpha(G)$ denote the clique and independence number of a graph G, respectively.)

Lemma 4 *Consider a set L of n lines in $\mathbb{R}^3$, such that no plane contains more than s lines, and no point is incident to more than ℓ lines of L. Let G be the*

[1] http://arxiv.org/abs/1512.03236

intersection graph of L. If $s, \ell \lesssim n^{1/2}$, then $\alpha(G) \gtrsim \sqrt{n}/\log \ell$. Moreover, if $r := \max\{s, \ell\} \gtrsim n^{\frac{1}{2}+\epsilon}$ for some $\epsilon > 0$, then $\alpha(G) \gtrsim n/r$.

Proof. If there is some regulus containing at least $n^{1/2}$ lines, we divide the lines into the two rulings of the regulus. One ruling contains at least half the lines, hence $\alpha(G) \gtrsim n^{1/2}$. We may therefore assume that the number of lines contained in a common regulus is at most $n^{1/2}$.

If $s, \ell \leq n^{1/2}$, the first term in the bound in Lemma 3 dominates, and applying Turán's Theorem gives $\alpha(G) \gtrsim \sqrt{n}/\log \ell$. If $r \geq n^{\frac{1}{2}+\epsilon}$, one of the latter terms dominates, and we apply Turán's Theorem to get $\alpha(G) \gtrsim n/r$. □

Proof. [Theorem 2] Suppose that such a graph G has $\alpha(G) \ll n^{1/3}$. Then by Lemma 4, $\max\{s, \ell\} \gtrsim n^{2/3}$. If $\ell \gtrsim n^{2/3}$ we are done, so $s \gtrsim n^{2/3}$. Therefore, we may assume that there is a plane containing $n^{2/3}$ lines. Divide these lines into classes of pairwise parallel lines. If some class contains at least $n^{1/3}$ lines, we have $\alpha(G) \gtrsim n^{1/3}$. Otherwise, there are at least $n^{1/3}$ distinct classes. Choosing one line from each class yields a clique of size $n^{1/3}$. □

Note that the Erdős-Hajnal property for intersection graphs of lines in $\mathbb{R}^3$ can be directly established from Alon et al. [1], but with a much smaller exponent. For $t = 3$, we also obtain a three-dimensional version of the dual of the result of Payne and Wood (Theorem 1 with $d = 2$).

Theorem 5 *Consider a collection L of n lines in $\mathbb{R}^3$, such that at most s lie in a plane, with $s \leq n/\log n$. Then there exists a point incident to $\sqrt{n}$ lines, or a subset of $\Omega(\sqrt{n})$ lines such that at most two intersect in one point.*

3 Stabbing lines in $\mathbb{R}^3$

Three lines in $\mathbb{R}^3$ are typically intersected by a fourth line, except in certain degenerate cases. Thus it makes sense to study configurations of lines in $\mathbb{R}^3$, and to consider a set of 4 or more lines degenerate if all its elements are intersected by another line. Here we provide a result for 6-tuples of lines.

We define a 6-tuple of lines to be degenerate if all six lines are intersected (or "stabbed") by a single line in $\mathbb{R}^3$. We call this line a *stabbing line* for the 6-tuple of lines. So in our framework this is the setting in which both $\mathcal{P}$ and $\mathcal{Q}$ are the set of lines in $\mathbb{R}^3$, and $t = 6$.

We make use of the Plücker coordinates and coefficients for lines in $\mathbb{R}^3$, which are a common tool for dealing with incidences between lines, see e.g. Sharir [17]. We prove the following Ramsey-type result for stabbing lines in $\mathbb{R}^3$.

Theorem 6 *Let L be a set of n lines in $\mathbb{R}^3$. Then either there is a subset of $\sqrt{n}$ lines of L that are all stabbed by one line, or there is a subset of $\Omega\left((n/\log n)^{1/5}\right)$ lines of L such that no 6-subset is stabbed by one line.*

Theorem 6 is an immediate consequence of the following generalisation of Theorem 1. The difference is that the set of hyperplanes $\mathcal{H}$ is arbitrary instead of being the set of all hyperplanes in $\mathbb{R}^d$. The proof is similar to that of Cardinal et al. [3].

Theorem 7 *Let $\mathcal{H}$ be a set of hyperplanes in $\mathbb{R}^d$. Then, every set of n points in $\mathbb{R}^d$ with at most ℓ points on any hyperplane in $\mathcal{H}$, where $\ell = O(n^{1/2})$, contains a subset of $\Omega\left((n/\log \ell)^{1/d}\right)$ points so that every hyperplane in $\mathcal{H}$ contains at most d of these points.*

We also have a simple construction for the following upper bound.

Theorem 8 *For every constant integer $t \geq 4$, there exists an arrangement L of n lines in $\mathbb{R}^3$ such that there is no subset of more than $O(\sqrt{n})$ lines that are all stabbed by one line, nor any subset of more than $O(\sqrt{n})$ lines with no t stabbed by one line.*

4 Lines and reguli in $\mathbb{R}^3$

Consider the case in which $\mathcal{P}$ is the class of lines in $\mathbb{R}^3$, $\mathcal{Q}$ is the class of reguli, and $t = 4$. Let P be a set of n lines, and assume that the lines in P are pairwise skew. Every triple of lines in P therefore determines a single regulus, and we may consider the set of all reguli determined by P. We consider the containment relation rather than intersection – we are interested in 4-tuples that all lie in the same regulus.

In order to prove our result, we first consider the case where $\mathcal{P} = \mathbb{R}^2$ and $\mathcal{Q}$ is a family of algebraic curves of bounded degree. We define the number of degrees of freedom of a family of algebraic curves $\mathcal{C}$ to be the minimum value s such that for any s points in $\mathbb{R}^2$ there are a constant number of curves passing through all of them. Moreover, $\mathcal{C}$ has multiplicity type r if any two curves in $\mathcal{C}$ intersect in at most r points. The proof of the following uses the Pach-Sharir bounds on the number of incidences between points and curves [15].

Theorem 9 *Consider a family $\mathcal{C}$ of bounded degree algebraic curves in $\mathbb{R}^2$ with constant multiplicity type and s degrees of freedom, for some $s > 2$. Then in any set of n points in $\mathbb{R}^2$, there exists a subset of $\Omega(n^{1-1/s})$ points incident to a single curve of $\mathcal{C}$, or a subset of $\Omega(n^{1/s})$ points such that at most s of them lie on a curve of $\mathcal{C}$.*

We now come back to our original question in which $\mathcal{P}$ is the class of lines in $\mathbb{R}^3$, $\mathcal{Q}$ is the class of reguli, and $t = 4$. Here we restrict the finite arrangement $P \subset \mathcal{P}$ to be pairwise skew, that is, pairwise nonintersecting and nonparallel. Recall that a regulus can be defined as a quadratic ruled surface which is the locus of all lines that are incident to three pairwise skew lines. There are only two kinds of reguli, both of which are quadrics – hyperbolic paraboloids and hyperboloids of one sheet [18].

Theorem 10 *Let L be a set of n pairwise skew lines in $\mathbb{R}^3$. Then there are $\Omega(n^{2/3})$ lines on a regulus, or $\Omega(n^{1/3})$ lines, no 4-subset of which lies on a regulus.*

The bounds can be shown to be tight.

Theorem 11 *There exists a set P of n pairwise skew lines in $\mathbb{R}^3$ such that there is no subset of more than $O(n^{2/3})$ lines on a regulus, and no more than $O(n^{1/3})$ lines such that no 4-subset lies on a regulus.*

Note that Aronov et al. [2] proved an upper bound on the number of incidences between lines and reguli in 3-space, from which one may derive an alternative proof of Theorem 10.

References

[1] Noga Alon, János Pach, Rom Pinchasi, Rados Radoicic, and Micha Sharir. Crossing patterns of semi-algebraic sets. *J. Comb. Theory, Ser. A*, 111(2):310–326, 2005.

[2] Boris Aronov, Vladlen Koltun, and Micha Sharir. Incidences between points and circles in three and higher dimensions. *Discrete & Computational Geometry*, 33(2):185–206, 2005.

[3] Jean Cardinal, Csaba D. Tóth, and David R. Wood. General Position Subsets and Independent Hyperplanes in d-Space. *ArXiv e-prints*, 2014, 1410.3637. To appear in *Journal of Geometry*.

[4] David Conlon, Jacob Fox, János Pach, Benny Sudakov, and Andrew Suk. Ramsey-type results for semi-algebraic relations. In *Proc. Symposium on Computational Geometry (SoCG)*, pages 309–318, 2013.

[5] David Conlon, Jacob Fox, and Benny Sudakov. Erdős-Hajnal-type theorems in hypergraphs. *J. Comb. Theory, Ser. B*, 102(5):1142–1154, 2012.

[6] György Elekes and Csaba D. Tóth. Incidences of not-too-degenerate hyperplanes. In *Proceedings of the 21st ACM Symposium on Computational Geometry (SoCG)*, pages 16–21, 2005.

[7] Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[8] Jacob Fox and János Pach. Erdős-Hajnal-type results on intersection patterns of geometric objects. *Bolyai Society Mathematical Studies – Horizon of Combinatorics*, 17:79–103, 2008.

[9] Jacob Fox and János Pach. Coloring K_k-free intersection graphs of geometric objects in the plane. *Eur. J. Comb.*, 33(5):853–866, 2012.

[10] Larry Guth and Nets H. Katz. Algebraic methods in discrete analogs of the Kakeya problem. *Advances Math.*, 225:2828–2839, 2010.

[11] Larry Guth and Nets H. Katz. On the Erdős distinct distances problem in the plane. *Annals Math.*, 181:155–190, 2015.

[12] Gyula Károlyi, János Pach, and Géza Tóth. Ramsey-type results for geometric graphs, I. *Discrete & Computational Geometry*, 18(3):247–255, 1997.

[13] Jan Kyncl. Ramsey-type constructions for arrangements of segments. *Eur. J. Comb.*, 33(3):336–339, 2012.

[14] David Larman, Jiri Matoušek, János Pach, and Jeno Torocsik. A Ramsey-type result for convex sets. *Bull. London Math. Soc.*, 26(2):132–136, 1994.

[15] János Pach and Micha Sharir. On the number of incidences between points and curves. *Combinatorics, Probability & Computing*, 7(1):121–127, 1998.

[16] Michael S. Payne and David R. Wood. On the general position subset selection problem. *SIAM J. Discrete Math.*, 27(4):1727–1733, 2013.

[17] Micha Sharir. On joints in arrangements of lines in space and related problems. *J. Comb. Theory, Ser. A*, 67.1:89–99, 1994.

[18] Micha Sharir and Noam Solomon. Incidences between points and lines on a two-dimensional variety. *ArXiv e-prints*, 2015, 1502.01670.

[19] Endre Szemerédi and William T. Trotter. Extremal problems in discrete geometry. *Combinatorica*, 3(3):381–392, 1983.

[20] Andrew Chi-Chih Yao and F. Frances Yao. A general approach to d-dimensional geometric queries (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–168, 1985.

Peeling the Cactus: Subexponential-Time Algorithms for Counting Triangulations*

Dániel Marx† Tillmann Miltzow‡

Abstract

Given a set of n points S in the plane, a triangulation T of S is a maximal set of non-crossing segments with endpoints in S. We present an algorithm that computes the number of triangulations on a given set of n points in time $n^{(11+o(1))\sqrt{n}}$, significantly improving the previous best running time of $O(2^n n^2)$ by Alvarez and Seidel [SoCG 2013]. Our main tool is identifying separators of size $O(\sqrt{n})$ of a triangulation in a canonical way. The definition of the separators are based on the decomposition of the triangulation into nested layers ("cactus graphs").

1 Introduction

Given a set of n points in the plane, a triangulation T of S is defined to be a maximal set of non-crossing line segments with both endpoints in S. This set of segments together with the set S defines a plane graph. It is easy to see that every bounded face of a triangulation T is indeed a triangle. Triangulations are one of the most studied concepts in discrete and computational geometry, studied both from combinatorial and algorithmic perspectives. It is well known that the number of possible triangulations of n points in convex position is exactly the $(n-2)$-th Catalan number, but counting the number of triangulations of arbitrary point sets seems to be a much harder problem. There is a long line of research devoted to finding better and better exponential-time algorithms for counting triangulations. The sequence of improvements culminated in the $O(2^n n^2)$ time algorithm of Alvarez and Seidel [2], winning the best paper award at SoCG 2013. Our main result significantly improves the running time of counting triangulations by making it subexponential:

Theorem 1 (General Plane Algorithm) *Given a set S of n points in the plane, there exists an algorithm that computes the number of all triangulations of S in $n^{(11+o(1))\sqrt{n}}$ time.*

*Supported by the ERC grant "PARAMTIGHT: Parameterized complexity and the search for tight complexity results", no. 280152.

†Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), dmarx@cs.bme.hu

‡Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), t.miltzow@gmail.com

It is very often the case that restricting an algorithmic problem to planar graphs allows us to solve it with much better worst-case running time than what is possible for the unrestricted problem. One can observe a certain "square root phenomenon": in many cases, the best known running time for a planar problem contains a square root in the exponent. For example, the 3-Coloring problem on an n-vertex graph can be solved in subexponential time $2^{O(\sqrt{n})}$ on planar graphs (e.g., by observing that a planar graph on n vertices has treewidth $O(\sqrt{n})$), but only $2^{O(n)}$ time algorithms are known for general graphs. Moreover, it is known that if we assume the Exponential-Time Hypothesis (ETH), which states that there is no $2^{o(n)}$ time algorithm for n-variable 3SAT, then there is no $2^{o(\sqrt{n})}$ time algorithm for 3-Coloring on planar graphs and no $2^{o(n)}$ time algorithm on general graphs [7]. The situation is similar for the planar restrictions of many other NP-hard problems, thus it seems that the appearance of the square root of the running time is an essential feature of planar problems. A similar phenomenon occurs in the framework of parameterized problems, where running times of the form $2^{O(\sqrt{k})} \cdot n^{O(1)}$ or $n^{O(\sqrt{k})}$ appear for many planar problems and are known to be essentially best possible (assuming ETH).

A triangulation of n points can be considered as a planar graph on n vertices, hence it is a natural question whether the square root phenomenon holds for the problem of counting triangulations. Indeed, for the related problem of finding a minimum weight triangulation, subexponential algorithms with running time $n^{O(\sqrt{n})}$ are known [4, 5]. These algorithms are based on the use of small balanced separators. Given a plane triangulation on n points in the plane, it is well known that there exists a balanced $O(\sqrt{n})$-sized separator that divides the triangulation into at least two independent graphs [6]. The basic idea is to guess a correct $O(\sqrt{n})$-sized separator of a minimum weight triangulation and recurse on all occurring subproblems. As there are only $n^{O(\sqrt{n})}$ potential graphs on $O(\sqrt{n})$ vertices, one can show that the whole algorithm takes $n^{O(\sqrt{n})}$ time [4, 5].

Unfortunately, this approach has serious problems when we try to apply it to counting triangulations. The fundamental issue with this approach is that a triangulation of course may have more than one $O(\sqrt{n})$-

sized balanced separators and hence we may overcount the number of triangulations, as a triangulation would be taken into account in more than one of the guesses. To get around this problem, an obvious simple idea would be to say that we always try to guess a "canonical" separator, for example, the lexicographically first separator. However, it is a complete mystery how to guarantee in subsequent recursion steps that the separator we have chosen is indeed the lexicographic first for all the triangulations we want to count.

Perhaps the most important technical idea of the paper is finding a suitable way of making the separators canonical. For this purpose, we define a decomposition of a triangulation into nested layers of cactus graphs. (A plane graph is a cactus graph if all vertices *and edges* are incident to the outer face.) The first layer is defined by the set of vertices and edges incident to the outer face. Inductively, the i-th layer is defined by the vertices and edges incident to the outer faces after the first $i-1$ layers are removed. Note that this definition may look similar to onion layers, but actually is very different. The outerplanar index of a graph is defined by the number of non-empty layers.

Given a triangulation T, we define small *canonical* separators by distinguishing two cases. If T has more than $\sqrt{n}$ cactus layers, then one of the first $\sqrt{n}$ layers has size at most $\sqrt{n}$ and we can define the one with smallest index to be the canonical separator. Using such a separator, we *peel off* some cacti to reduce the problem size. In the case when we have only a few cactus layers, we can define short canonical separator paths from the interior to the outer face of the triangulation. We formalize both ideas into a dynamic programming algorithm. The main difficulty is to define the subproblems appropriately. We use the so-called ring subproblems for the layer separators and ring sector subproblems for the path separators.

As a byproduct of this algorithmic scheme, we can efficiently count triangulations with a small number of layers. This is similar to previous work on finding a minimum weight triangulation [3] and counting triangulations [1] for point sets with a small number of onion layers.

Theorem 2 (Thin Plane Algorithm) *Given a set S of n points in the plane, there exists an algorithm that computes the number of all triangulations of S with outerplanar index k in $n^{O(k)}$ time.*

2 Ring Subproblems

Our algorithm is based on dynamic programming: we define a large number of subproblems that are *more general* than the problem we are trying to solve. We generalize the problem by considering *rings:* we need to triangulate a point set in a region between two polygons. Additionally, we may have layer constraints prescribing that a certain number of vertices should appear on certain layers.

In this section, we give a vague definition of the ring subproblems used by the algorithm and sketch how an algorithm that can solve those problems implies Theorem 1 for counting triangulations. In Section 3 we will sketch how to solve these subproblems for "thin rings". Section 4 sketches how to solve ring subproblems in full generality using the algorithm from Section 3 as an important subroutine.

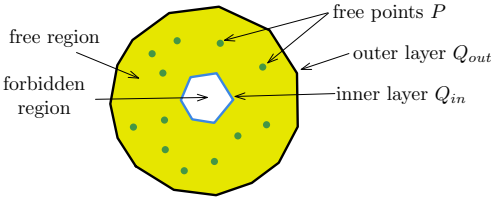


Figure 1: A simple ring subproblem.

See Figure 1 for an illustration of the following definition. A *ring subproblem* consists of: an outer layer Q_{out}, which consists of one or more simple polygons (a ring subproblem is *simple* in case that there is only one polygon.); an inner layer Q_{in}, which is a cactus graph and potentially empty; an inner and outer layer index, which serve to determine the width and conveniently combine solutions of ring subproblems; and potentially layer-constraints that indicate the size of each layer. A *valid triangulation* of a ring subproblem is a triangulation of the area and points between the inner and outer layer, which satisfies the width conditions and in case that layer-constraints are present: each layer should have an appropriate size.

Theorem 3 *Given a layer-unconstrained ring subproblem $\mathcal{S}$ with n free points, there exists an algorithm, denoted by* GeoRing, *that computes the number of all triangulations of $\mathcal{S}$ in $n^{(11+o(1))\sqrt{n}}$ time.*

Proof. [Sketch Theorem 1] The way, we use Theorem 3 is to define for each k an layer-unconstrained ring subproblem $\mathcal{O}_k$ such that each k-outerplane triangulation of S corresponds to a valid k-outerplane triangulation of $\mathcal{O}_k$ and vice versa. Then the algorithm to count all triangulations of S is to count all triangulations of $\mathcal{O}_k$, for each $k = 1, \ldots, n$. It takes $n^{O(\sqrt{n})}$ time for each k. We define $\mathcal{O}_k$ as follows. The outer layer Q_{out} is the boundary of the convex hull of S. The inner layer Q_{in} is empty. The free points P are S without the points on the boundary of the convex hull of S. The inner and outer layer index are in-index$(\mathcal{O}_k) = k + 1$ and out-index$(\mathcal{O}_k) = 1$. □

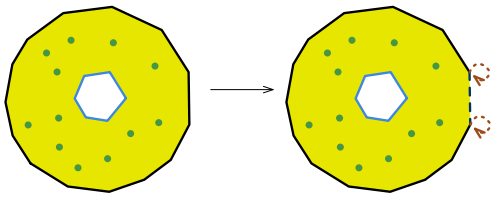


Figure 2: left: a simple ring subproblem right: the transformed ring sector subproblem.

3 Thin Rings

This section is devoted to the proof of the following theorem, which gives an algorithm for solving ring subproblems with a certain width w. This algorithm will be invoked by the main algorithm for values $w \leq \sqrt{n}$. We will sketch the main ideas of the algorithm and its runtime analysis.

Theorem 4 *Given a simple (layer-constrained or layer-unconstrained) ring subproblem $\mathcal{S}$ with width w, there exists an algorithm* RingSec *that computes the number of all valid triangulations of $\mathcal{S}$ in time $n^{(5+o(1))w}$.*

Theorem 4 implies easily Theorem 2 in a similar fashion as Theorem 3 implies Theorem 1.

We use a different kind of separator for this algorithm. This requires a yet more specialized definition of subproblems for our dynamic programming scheme: *ring sector subproblems*. We sketch the proof of Theorem 5. It easily implies Theorem 4, using a simple transformation from ring subproblems to ring sector subproblems, see Figure 2.

Theorem 5 *Given a ring sector subproblem $\mathcal{S}$ with width w on a set of n points, there exists an algorithm, denoted* RingSec *that computes the number of all valid triangulations of $\mathcal{S}$ in $n^{(5+o(1))w}$ time.*

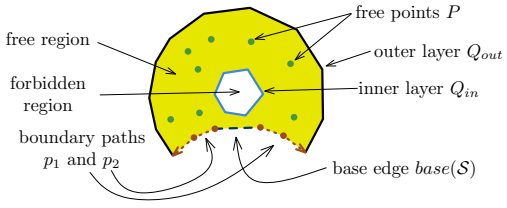


Figure 3: A ring sector subproblem.

Ring Sector subproblems are defined very similar to ring subproblems, see Figure 3. The essential difference is that some part of the outer layer is removed and replaced by two boundary paths and a base edge. These components are introduced for the recursion step as illustrated in Figure 4. A *valid triangulation* of a ring sector subproblem needs to satisfy some additional boundary constraints to ensure the boundary paths are indeed canonical separators of all counted triangulations.

Given a valid triangulation T of a ring sector problem $\mathcal{S}$ it is not hard to show that every vertex on layer i has a neighbor w.r.t. T in layer $i-1$. Furthermore, there is a unique triangle Δ incident to the base edge. From the vertex v of Δ that is not incident to the base edge, exists a path to the outer layer of $\mathcal{S}$ by always choosing an adjacent vertex closer to the outer layer. There is *exactly one* such path p, if we further require that the vertex with lowest order label is taken. (The order label is some distingued number from {1,...,n} that was fixed in advance for each vertex.) Such paths are called *canonical outgoing paths*. We recurse on a ring sector subproblem by guessing all potential such triangles Δ and all potential canonical paths p as described above. For each such path, we can define two subproblems $\mathcal{S}_{\text{right}}$ and $\mathcal{S}_{\text{left}}$, see Figure 4. We can

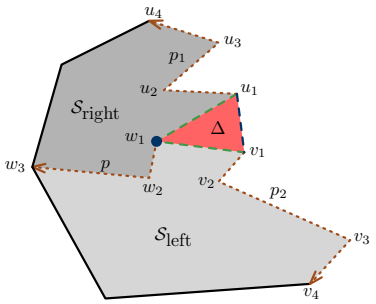


Figure 4: Splitting a ring sector subproblem.

restrict our triangulation T to these subproblems and receive two new triangulations T_{left} and T_{right}, and conversely, given two triangulations T_{left} and T_{right}, we can combine it to a triangulation T. Thus if we recursively count the number of valid triangulations of subproblems $\mathcal{S}_{\text{right}}$ and $\mathcal{S}_{\text{left}}$, then we get *exactly* the number of valid triangulations of $\mathcal{S}$ where Δ is the triangle incident to the base edge and p is the canonical outgoing path starting at vertex v or Δ. Summing up for every possible triangle Δ and path p, we get exactly the number of valid triangulations of $\mathcal{S}$.

If there are layer constraints in $\mathcal{S}$, then we have to do some more work. Let d, d_{left}, and d_{right} be the vectors that indicate the size of the layers for T, T_{left}, and T_{right} respectively. Except for the vertices shared by T_{left} and T_{right}, it holds that d equals $d_{\text{left}} + d_{\text{right}}$. Now, let us go back to our subproblems $\mathcal{S}_{\text{left}}$ and $\mathcal{S}_{\text{right}}$. Let c be the layer-constraint vector of $\mathcal{S}$. Then, we define all pairs of compatible layer-constraints $(c_{\text{left}}, c_{\text{right}})$ such that two valid triangulations for $\mathcal{S}_{\text{right}}(c_{\text{right}})$ and $\mathcal{S}_{\text{left}}(c_{\text{left}})$ respectively give a triangulation for $\mathcal{S}$ with the correct number of vertices on each layer. Here, the technical difficulty is

to take into consideration the vertices shared by both subproblems. Further we need to ensure that vertices in the i-th layer of $\mathcal{S}_{\text{right}}$ will also be in the i-th layer of $\mathcal{S}$. We recurse on all subproblems occurring in this way.

Proof. [Sketch Theorem 5] The correctness of the algorithm follows from the correctness of the recursion. The bound on the running time follows from bounding the time required to solve a subproblem times the number of subproblems. We save our intermediate results in a search tree in order to prevent to handle any subproblem more than once. The bound on the number of subproblems follows from the fact that all of their components are defined by at most two path separators, and from the fact that a separator has at most length w there are at most $n^{O(w)}$ of them. The number of layer constraints is bounded by the assumption that at most $\sqrt{n}$ layers are non-zero. The time for the recursive steps for one subproblem can be asymptotically bounded by the number of recursions, which in turn depends only on the number of potential canonical paths and ways to split the layer constraints in a compatible way. □

4 General Ring Subproblems

We briefly sketch the main algorithm in this section and estimate its running time.

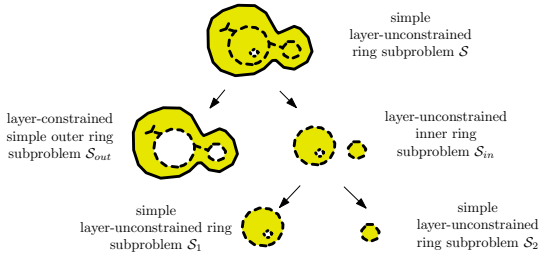


Figure 5: Overview of the algorithm.

The way we solve general ring subproblems is to distinguish two cases. In the case that the ring subproblem is thin, that is, has only few layers ($\leq \sqrt{n}$), we will use the algorithm of Theorem 4 as explained in Section 3. In case that we have many layers ($> \sqrt{n}$), we know that one of the outermost $\sqrt{n}$ layers must be of size $\leq \sqrt{n}$ by the pigeon hole principle. We use this layer as a separator that splits the problem into a thin outer part and an inner part.

To be more explicit, let $\mathcal{S}$ be a ring problem and T be valid triangulation of $\mathcal{S}$. By the outer and inner layer index of $\mathcal{S}$, we already know exactly the number of layers that T has. Consider the case that T has more than $\sqrt{n}$ layers. Then among the $\sqrt{n}$ layers closest to the outer layer, one must have size less than or equal to $\sqrt{n}$. Note that the layer L that is actually closest to the outer layer of $\mathcal{S}$ is *uniquely* determined. We try to guess this layer L, which requires guessing at most $\sqrt{n}$ points. The guess defines an inner ring subproblem $\mathcal{S}_{\text{in}}$ and an outer ring subproblem $\mathcal{S}_{\text{out}}$, as depicted in Figure 5. In case the cactus layer L is disconnected or has disconnected bounded faces, the inner ring subproblems consist of several components. In this case, we split it into smaller subproblems, before we proceed with the recursion.

We can restrict T to $\mathcal{S}_{\text{out}}$ to attain a triangulation T_{out}. It is clear that all layers different from L in T_{out} have size larger than $\sqrt{n}$. Therefore, we want to count only those triangulations of $\mathcal{S}_{\text{out}}$ that have all layers (except L) of size larger than $\sqrt{n}$. We use layer constraints for this purpose: we solve $\mathcal{S}_{\text{out}}$ with every possible layer constraint where every layer is required to have size greater than $\sqrt{n}$.

The running time can be estimated by bounding the total number of ring subproblems times the time spent per ring subproblem. Each ring subproblem is defined by an inner and outer layer and a layer constraint. In the course of the algorithm only inner and outer layers of size less than or equal to $\sqrt{n}$ are guessed, and there are at most $n^{O(\sqrt{n})}$ of them. As we will never constrain more than $\sqrt{n}$ layers to be non-zero, the total number of layer constraints is bounded by $n^{O(\sqrt{n})}$. For the case of rings with width smaller than $\sqrt{n}$, the runtime is given by Theorem 4. In the other case, the bound stems from the total number of layers of size $\sqrt{n}$, which is $n^{O(\sqrt{n})}$.

References

[1] V. Alvarez, K. Bringmann, R. Curticapean, and S. Ray. Counting triangulations and other crossing-free structures via onion layers. *D&C*, 53(4):675–690, 2015.

[2] V. Alvarez and R. Seidel. A simple aggregative algorithm for counting triangulations of planar point sets and related problems. In *SoCG'13*, pages 1–8, 2013.

[3] E. Anagnostou and D. Corneil. Polynomial-time instances of the minimum weight triangulation problem. *Computational Geometry*, 3(5):247–259, 1993.

[4] C. Knauer and A. Spillner. A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In *WG'06*, pages 49–57, 2006.

[5] A. Lingas. Subexponential-time algorithms for minimum weight triangulations and related problems. In *CCCG'98*, 1998.

[6] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[7] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

Holes in 2-convex point sets*

Oswin Aichholzer† Martin Balko‡ Thomas Hackl† Alexander Pilz§ Pedro Ramos¶
Birgit Vogtenhuber† Pavel Valtr‡

Abstract

Let S be a finite set of n points in the plane in general position. A *k-hole* of S is a simple polygon with k vertices from S and no points of S in its interior. A simple polygon P is *l-convex* if no straight line intersects the interior of P in more than l connected components. Moreover, a point set S is *l-convex* if there exists an l-convex polygonalization of S.

Considering a typical Erdős-Szekeres type problem we show that every 2-convex point set of size n contains a convex hole of size $\Omega(\log n)$. This is in contrast to the well known fact that there exist general point sets of arbitrary size that do not contain a convex 7-hole. Further, we show that our bound is tight by providing a construction for 2-convex point sets with holes of size at most $O(\log n)$.

1 Introduction

Let S be a set of n points in the plane in general position, i.e., S does not contain a collinear point triple. A *k-hole* of S is a simple polygon whose k vertices are a subset of S and whose interior does not contain any point of S. Erdős [4] asked for the smallest integer $h(k)$ such that every set of $h(k)$ points in the plane contains at least one convex k-hole. Here, we consider this question for a restricted class of point sets.

A simple polygon P with boundary ∂P is *l-convex* if there exists no straight line that intersects the interior of P in more than l connected components [1]. We call a line that intersects ∂P in a finite set of at least j points a *j-stabber*; for an l-convex polygon, there cannot be a $(2l+1)$-stabber. Clearly, a convex polygon is 1-convex. In [2], the notion of l-convexity was transcribed to finite point sets. A point set S is *l-convex* if there exists a polygonalization $P(S)$ of S such that $P(S)$ is an l-convex polygon. Note that an l-convex polygon or point set is also $(l+1)$-convex. In this paper, we consider the following problem: What is the smallest number $f(k)$ such that any 2-convex point set of size $f(k)$ contains a convex k-hole?

Similar problems (for different generalizations of convexity) have also been considered, see e.g. [7, 8]. It has been shown that $h(k)$ is finite for $k \leq 6$, see e.g. [3] for details. For general point sets Horton [6] showed that there exist sets of arbitrary size that do not contain a convex 7-hole, that is, $h(7)$ is not bounded. In contrast we show that every 2-convex point set of size n contains a convex hole of size $\Omega(\log n)$, implying that $f(k)$ is bounded for any $k > 0$ (Section 3). Further, we show that our bound is tight by providing a construction for 2-convex point sets with holes of size at most $O(\log n)$ (Section 4). Due to space constraints, most proofs are omitted.

2 Properties of 2-convex polygons

We follow the definitions used in [1] and [2]. A *pocket* of a simple polygon P is a maximal chain on the boundary of P not containing any vertices of $\mathrm{CH}(P)$ except for its endpoints. For 2-convex polygons, the following is known about the structure of the pockets.

Lemma 1 ([1], Lemma 12) *Let $K = \langle p_0, \ldots, p_t \rangle$ be a pocket of a 2-convex polygon between two extreme points p_0 and p_t. Then K can be partitioned into three chains $C_1 = \langle p_0, p_1, \ldots, p_r \rangle$, $C_2 = \langle p_{r+1}, \ldots, p_s \rangle$, and $C_3 = \langle p_{s+1}, \ldots, p_t \rangle$ for $0 \leq r \leq s < t$, such that all vertices in C_1 and C_3 are convex vertices of P, while all vertices in C_2 are reflex.*

We call the segment p_0p_t the *lid* of the pocket. If C_2 is empty, the pocket consists solely of a convex hull edge. Otherwise, we call the edges p_rp_{r+1} and p_sp_{s+1} the two *inflection edges* of the pocket. Consider the (convex) polygons defined by C_1, C_2, and C_3, respectively. The next lemma follows from the proof of Lemma 12 in [2].

Lemma 2 ([2]) *The interior of a convex polygon defined by C_1, C_2, or C_3 does not intersect ∂P.*

*Research supported by OEAD project CZ 18/2015 and by project no. 7AMB15A T023 of the Ministry of Education of the Czech Republic. O.A., A.P., and B.V. supported by ESF EUROCORES programme Euro-GIGA - ComPoSe, Austrian Science Fund (FWF): I648-N18. M.B. and P.V. supported by grant GAUK 690214 and by project CE-ITI no. P202/12/G061 of the Czech Science Foundation GAČR. T.H. supported by Austrian Science Fund (FWF): P23629-N18.

†Institute of Software Technology, Graz University of Technology. [oaich|thackl|bvogt]@ist.tugraz.at.

‡Department of Applied Mathematics and Institute for Theoretical Computer Science (ITI), Charles University, [balko|valtr]@kam.mff.cuni.cz.

§Institute of Theoretical Computer Science, ETH Zürich, alexander.pilz@inf.ethz.ch.

¶Departamento de Física y Matemáticas, Universidad de Alcalá, pedro.ramos@uah.es.

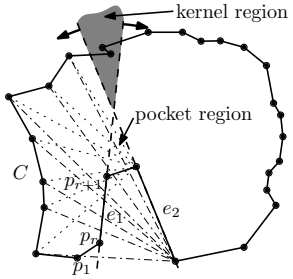


Figure 1: The order of the vertices defined by the inflection edges of a pocket ([2, Figure 9], relabeled). The gray wedge is the kernel region.

Lemma 3 ([2], Lemma 10) *Let P be a 2-convex polygon and let e_1 and e_2 be the inflection edges of a pocket K directed from the convex to the reflex vertex, with the vertices defined as in Lemma 1. Without loss of generality, p_r is left of e_2, i.e., $e_1 = p_r p_{r+1}$ and $e_2 = p_{s+1}p_s$. Let C be the part of ∂P defined by the vertices that are to the left of e_2 and not part of the pocket (starting at p_1, the left endpoint of the lid of K). Then the order of the points in C along ∂P is the same as the radial order around any point p on e_2. An analogous statement holds for any point on e_1 and the points of ∂P to the right of e_1.*

See Figure 1 for an illustration (taken from [2, Figure 9]). The *kernel region* of the pocket K with non-empty C_2 is the region that is to the left of e_1, to the right of e_2, and, if $r+1 \neq s$, to the left of $p_{r+1}p_s$. Observe that, for a star-shaped 2-convex polygon, the kernel of the polygon is the intersection of the kernel regions of all the pockets.

3 The lower bound

Let S be a 2-convex point set in the plane in general position and let P be a 2-convex polygon that is a polygonalization of S. In this section, we prove the following.

Theorem 4 *Every 2-convex point set of size n contains a convex k-hole for $k \in \Omega(\log n)$.*

Let us first sketch the proof: If P has a large pocket, Lemma 2 implies the existence of a large k-hole. When P has no large pocket, we will use Lemma 5 to find a large set $Q \subset S$ of points in convex position. If Q forms a hole in S, we are done. Finally, if Q does not form a hole in S, we will use Lemma 7 and Lemma 10 to find a big enough convex hole.

Lemma 5 *Let m be the size of the largest pocket in S. Then there exists a point p (probably not in S) s.t. there is a sequence σ of $\lceil \frac{n}{3m} \rceil - 1$ points of S that are separated by a line from p, and their order around p matches the order along ∂P, where they appear consecutively.*

Proof. Suppose first that P is star-shaped and let $p \notin S$ be a point in the kernel of P. Consider any half-plane H defined by a line through p that contains $\lceil \frac{n}{2} \rceil$ points of S. The radial order of the points in $S \cap H$ around p is the same as the order along P.

Suppose now that P is not star-shaped, i.e., its kernel is empty. The kernel of P is determined by the intersection of the kernel regions of all the pockets. A non-empty kernel region is the intersection of two half-planes defined by inflection edges (as discussed in [2]). By Helly's theorem [5], we know that, if the kernel of P is empty, there exists a triple of inflection edges such that the intersection of the half-planes (partly) defining their kernel regions is empty. (Similar to [2, Lemma 11].) This means that there exists at least one inflection edge e of a pocket K such that the open half-plane H defined by e that contains K also contains at least $\lceil n/3 \rceil$ points of S. Due to Lemma 3, the radial order of the points in $S \cap H$ and not on K around any point p on e is the same as their order along ∂P. Hence, there is a sequence of at least $\left\lceil \frac{\lceil n/3 \rceil - (m-2)}{m-2} \right\rceil \geq \lceil \frac{n}{3m} \rceil - 1$ points along ∂P that are consecutive in the order of all points of S around p (not containing a point of K and linearly separated from p by the supporting line of an edge of K). □

In the previous proof, when P is star-shaped, the point p was not part of S. However, we can define a point set S' consisting of p and $S \cap H$. Then, it is easy to see that there is a 2-convex polygonization P' of S' in which p sees all the points in the order as they appear along $\partial P'$. Any convex k-hole of S' is a convex $(k-1)$-hole or a convex k-hole of S. Thus, for simplicity, we will assume that $p \in S$.

Let $\phi \subseteq S^3$ be the ternary relation representing the cyclic order of the vertices of P as they appear on the boundary of P traversed in counterclockwise direction. That is, a triple (u, v, w) of points of S is in ϕ if we can trace u, v, w in this order along the boundary of P in counterclockwise direction. For $u, w \in S$, a *(closed) interval $[u, w]$ from u to w* in ϕ is the set $\{v \in S \colon (u, v, w) \in \phi\} \cup \{u, w\}$. Note that the intervals $[u, w]$ and $[w, u]$ are in general distinct. Each point $u \in S$ defines a linear order $<_u$ on $S \setminus \{u\}$ where $x <_u y$ if and only if $(x, y, u) \in \phi$.

Note that vertices of a pocket $K = \langle p_0, \ldots, p_t \rangle$ of P induce a closed interval $[p_0, p_t]$ in ϕ. Consequently, ϕ induces a cyclic order of pockets of P. We choose an arbitrary pocket K_0 of P and use $K_0, \ldots, K_{m-1}$ to denote this cyclic order where m is the number of pockets of P. In the rest of the section, the indices of pockets are always taken modulo m.

For $r, s \in \{0, \ldots, m-1\}$, we use $[K_r, K_s]$ to denote the interval consisting of pockets $K_r, K_{r+1}, \ldots, K_s$.

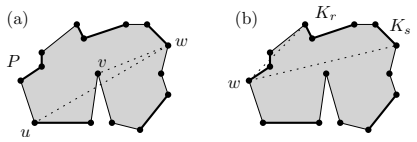


Figure 2: (a) An example of a reversed triple (u, v, w). (b) The point w controls the interval $[K_r, K_s]$.

The *length* of $[K_r, K_s]$ is the number of pockets in $[K_r, K_s]$. A *subinterval* of $[K_r, K_s]$ is any interval that can be obtained from $[K_r, K_s]$ by deleting the first i and the last j consecutive pockets of $[K_r, K_s]$ for some $i, j \in \mathbb{N}_0$.

We say that a triple $(u, v, w) \in \phi$ is *reversed* if the triangle with the vertices u, v, w traced in this order is oriented clockwise.

For an interval $[K_r, K_s]$, a point v from $S \setminus (\cup_{i=r-1}^{s+1} K_i)$ *controls* $[K_r, K_s]$ if the following conditions are satisfied:

(i) There is no reversed triple (x, y, v) with x and y contained in distinct pockets of $[K_r, K_s]$,

(ii) $\mathrm{CH}(\cup_{i=r}^{s} K_i)$ contains no point of $S \setminus (\cup_{i=r}^{s} K_i)$,

(iii) $\mathrm{CH}(\cup_{i=r}^{s} K_i \cup \{v\})$ contains no point of $S \setminus (\cup_{i=r}^{s} K_i)$ except of vertices of pockets containing v.

We note that Condition (i) especially implies that there is no reversed triple (x, y, v) with x and y being vertices of pockets in $[K_r, K_s]$ and x or y being a convex hull vertex. Hence, if v controls $[K_r, K_s]$, then v also controls every subinterval of $[K_r, K_s]$. Further, Condition (i) implies that v is linearly separable from $[K_r, K_s]$.

Lemma 6 *Let (u, v, w) be a reversed triple of points in S and let ab be the lid of the pocket K of v s.t. $(a, v, b) \in \phi$. If $\overline{uw}$ separates v from ab, then the order $<_v$ is the same as the radial order around v for $[u, a]$ and for $[b, w]$.*

Proof. We prove the statement for $[u, a]$, as the argument for $[b, w]$ follows by symmetry. Let C be the part of ∂P defined by the interval $[u, a]$. Since $\overline{uw}$ separates v from ab and thus intersects K twice, its only intersection with C is at u. Hence, any line through v crossing C has exactly one ray starting at v crossing C. Suppose there exists a line ℓ through v s.t. the ray r crossing C has a crossing with C where it enters P. We claim that a perturbation of ℓ is a 6-stabber of P, contradicting 2-convexity. Let r' be the complement of r on ℓ.

Suppose first that r enters the interior of P at v. Then r intersects ∂P in at least three points other than v. Since ab is separated from v by $\overline{uw}$, r' crosses ∂P in a point not on the pocket K. Thus, if r' leaves P at v, then ℓ is a 6-stabber. If r' does not leave P at v, then ℓ supports ∂P at v, in which case there is a perturbation of ℓ that is a 6-stabber.

Suppose now that r leaves P at v. Since ab is an edge of the convex hull of S and r crosses C, r cannot cross ab. Hence, it enters P again at the pocket K, implying that there are at least four points other than v where r crosses ∂P. The fact that r' intersects ∂P in a point not on C makes ℓ a 6-stabber.

Therefore, there is no ray starting at v entering P at C, which completes the proof. □

Lemma 7 *Let K_i, K_j, and K_l be pockets in a sequence of pockets that is controlled by a point $p \in S$. Let (u, v, w) be a reversed triple of points from S such that u, v, and w are contained in K_i, K_j, and K_l, respectively. Then v controls the intervals $[K_{i+1}, K_{j-2}]$ and $[K_{j+2}, K_{l-1}]$, provided that $\overline{uw}$ separates v from the endpoints of K_j.*

Lemma 8 *Let $[K_r, K_{r+3d+3}]$ be an interval controlled by some point $p \in S$. Then there is a subinterval of $[K_r, K_{r+3d+3}]$ of length d controlled by a point of a pocket that is contained in $[K_r, K_{r+3d+3}]$.*

Let H be a hole in S. If H contains at most one point from every pocket of S, then H is *transversal*. We say that an interval $[K_r, K_s]$ of pockets *contains a hole H* if every vertex of H is contained in some pocket of the interval $[K_r, K_s]$. We call a hole H *nice*, if there is no reversed triple of vertices of H.

Lemma 9 *For every integer $k \geq 2$, let $[K_r, K_s]$ be an interval of pockets that contains a nice convex transversal $(k-1)$-hole. If a point p of S controls $[K_r, K_s]$, then there is a pocket K containing p such that the intervals $[K_r, K]$ and $[K, K_s]$ contain a nice convex transversal k-hole.*

First, we prove the following lemma and then we show how it implies Theorem 4.

Lemma 10 *For every positive integer k and every interval $[K_r, K_s]$ of pockets, if the length of $[K_r, K_s]$ is at least $2 \cdot 3^k - 2$ and $[K_r, K_s]$ is controlled by some point of S, then $[K_r, K_s]$ contains a nice convex transversal k-hole.*

Proof. We prove the lemma by induction on k. For $k = 1$, the lemma follows from the fact that every interval of length 1 contains a 1-hole. For the induction step, let $k > 1$. For $d := 2 \cdot 3^{k-1} - 2$, let $[K_r, K_s]$ be the interval of length at least $3d + 4 = 2 \cdot 3^k - 2$ that is controlled from some point of S. By Lemma 8, there is a point q contained in a pocket from $[K_r, K_s]$ such that q controls a subinterval $[K_i, K_j]$ of $[K_r, K_s]$ with length at least d. Using the induction hypothesis, it follows that $[K_i, K_j]$ contains a nice convex transversal

$(k-1)$-hole H. By Lemma 9, the hole H can be extended to a nice convex transversal k-hole contained in $[K_r, K_s]$. □

Proof of Theorem 4. To show that Lemma 10 implies Theorem 4, we prove that in every 2-convex point set S of size n there is a convex k-hole for $k \geq \log n/3$, or we have an interval of length $\Omega(n/\log^3 n)$ that is controlled by a point from S. In the latter case we then apply Lemma 10 and obtain a convex k-hole with $k \geq c \log n$ for an absolute constant $c > 0$.

First, assume that there is a pocket $K = \langle p_0, \ldots, p_t \rangle$ in P with $t \geq \log n$ in P. By Lemma 1, the pocket K can be partitioned into three chains $C_1 = \langle p_0, p_1, \ldots, p_r \rangle$, $C_2 = \langle p_{r+1}, \ldots, p_s \rangle$, and $C_3 = \langle p_{s+1}, \ldots, p_t \rangle$ for $0 \leq r \leq s < t$, such that all vertices in C_1 and C_3 are convex in P, while all vertices in C_2 are reflex. Since K contains at least $\log n$ vertices, at least one of the chains C_1, C_2, and C_3 contains at least $\log n/3$ vertices. For some $i \in \{1, 2, 3\}$, let C_i be such a chain. By Lemma 2, the vertices of C_i are vertices of a convex k-hole for $k \geq \log n/3$. See Figure 3 (a).

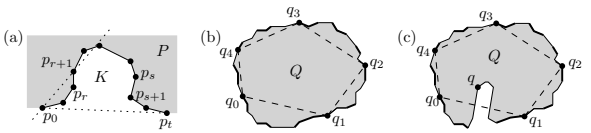


Figure 3: (a) A large pocket gives a large hole. (b) If no point of S interferes, then Q is a hole. (c) If there is a point inside Q, then we use Lemma 7 and apply Lemma 10.

In the rest of the proof we thus assume that every pocket of P contains less than $\log n$ vertices. In particular, there are more than $n/\log n$ pockets in P and $\mathrm{CH}(S)$ has more than $n/\log n$ vertices. By Lemma 5, there are at least $m := \left\lceil \frac{n}{3 \log n} \right\rceil - 1$ points that are "controlled" by a point p (that is not necessarily in S). We call these points the *initial interval*. However, by the discussion after Lemma 5 we can assume for the following that $p \in S$. Let $q_0, \ldots, q_{\log n - 1}$ be vertices of $\mathrm{CH}(S)$ traced in counterclockwise direction along the boundary of P in the initial interval such that the points in each interval $[q_i, q_{i+1}]$ for $i = 0, \ldots, \log n - 1$ (indices taken modulo $\log n$) form at least $m/\log^2 n$ pockets. Clearly, if the polygon Q with the vertices $q_0, \ldots, q_{\log n - 1}$ is a hole, then we are done; see Figure 3 (b). Otherwise there is a point q in the interior of Q and we have a reversed triple (q_i, q, q_j) for some $i, j \in \{0, \ldots, \log n - 1\}$. Let K, K', and K'' be pockets containing q_i, q, and q_j, respectively. The endpoints of K' are separated from q by $\overline{q_i q_j}$, as q_i and q_j are vertices of $\mathrm{CH}(S)$; See Figure 3 (c). By Lemma 7, the point q controls the interval of pockets that are between K and K' and between K' and K''. From the choice of Q, at least one of these intervals has length at least $m/(2\log^2 n) = \Omega(n/\log^3 n)$. □

4 An upper-bound construction

Theorem 11 *For any n there exists a 2-convex point set S of size n such that all convex holes it contains have size $O(\log n)$.*

Proof. The set is constructed recursively, following the idea shown in Figure 4. We define $S_i = L_i \cup R_i \cup \{c_i\}$, where L_i and R_i are flattened enough copies of S_{i-1}. For $i = 0$, we set $L_0 = R_0 = \emptyset$.

An empty convex hole K intersecting R_i cannot intersect both the left and right part of L_i, and this is true for every level in the recursion. Of course, an analogus statement is true if K intersects L_i. Therefore, $|K| = O(\log n)$. □

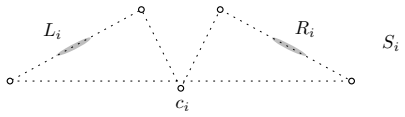


Figure 4: Recursive operation for the construction of an upper bound example.

Acknowledgments. This work was initiated during the ComPoSe Workshop on Algorithms using the Point Set Order Type held in March/April 2014 in Ratsch, Austria.

References

[1] O. Aichholzer, F. Aurenhammer, E. D. Demaine, F. Hurtado, P. Ramos, and J. Urrutia. On k-convex polygons. *Comput. Geom.*, 45(3):73–87, 2012.

[2] O. Aichholzer, F. Aurenhammer, T. Hackl, F. Hurtado, A. Pilz, P. Ramos, J. Urrutia, P. Valtr, and B. Vogtenhuber. On k-convex point sets. *Comput. Geom.*, 47(8):809–832, 2014.

[3] O. Aichholzer, R. Fabila-Monroy, H. Gonzalez-Aguilar, T. Hackl, M. A. Heredia, C. Huemer, J. Urrutia, P. Valtr, and B. Vogtenhuber. On k-gons and k-holes in point sets. *Comput. Geom.*, 48(7):528–537, 2015.

[4] P. Erdős. Some more problems on elementary geometry. *Austral. Math. Soc. Gaz.*, 5:52–54, 1978.

[5] E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresber. Deutsch. Math.-Verein.*, 32:175–176, 1923. In German.

[6] J. D. Horton. Sets with no empty convex 7-gons. *Canad. Math. Bull.*, 26(4):482–484, 1983.

[7] P. Valtr. A sufficient condition for the existence of large empty convex polygons. *Discrete Comput. Geom.*, 28(4):671–682, 2002.

[8] P. Valtr, G. Lippner, and Gy. Károlyi. Empty convex polygons in almost convex sets. *Period. Math. Hungar.*, 55(2):121–127, 2007.