University *of Ljubljana*
Faculty *of Computer and Information Science*

University *of Ljubljana*
Faculty *of Mathematics and Physics*

EUROPEAN SCIENCE FOUNDATION

European Workshop on Computational Geometry

EuroCG 2015

March 15-18, Ljubljana, Slovenia

BOOK OF ABSTRACTS

http://eurocg15.fri.uni-lj.si

Preface

Dear reader,

here are the abstracts of works presented at the 31st European Workshop on Computational Geometry (EuroCG 2015) held on March 15-18, 2015 in Ljubljana, Slovenia. The event was hosted by the University of Ljubljana, Faculty of Computer and Information Science.

The EuroCG is an annual, informal workshop whose goal is to provide a forum for scientists to meet, present their work, interact, and establish collaborations, in order to promote research in the field of Computational Geometry. The workshop aims at providing an informal atmosphere where established and young researchers have a possibility for a productive exchange of ideas and collaboration.

EuroCG does not have formally reviewed proceedings, although the contributions are reviewed and certain improvements to authors are suggested by a Programme Committee. This volume contains a collection of **64** abstracts of talks presented at the workshop. The abstracts should be regarded as preprints, and therefore results presented at EuroCG are often also submitted to peer-reviewed conferences and journals.

There were 77 submissions made to EuroCG 2015, and two of them were not considered because of their format. Each remaining submission was reviewed by at least two members of a Programme Committee. Most of the work of the Program Committee was done through EasyChair. The Program Committee finally decided to accept 66 papers for presentation. Since authors of two submissions were not able to present their papers the final number of papers was 64.

Besides the contributing talks, we also had 3 invited talks, delivered by Aleš Leonardis, Kurt Mehlhorn, and Bojan Mohar. A very short description is also included in this volume.

Such a meeting requires the effort of a lot of parties. We would like to thank the authors for submitting their abstracts and the members of the Program Committee for their work on selecting the papers. We thank EasyChair for making its valuable platform available for free. Next, we thank European Science Foundation (ESF) under the EUROCORES Programme EuroGIGA for their support that lead to a substantial reduction in the registration fees. Finally, we would like to thank the members of the Organizing Committee for their work to make the event as smooth as possible.

March 2015
Ljubljana

Andrej Brodnik and Sergio Cabello

Program Committee

Prosenjit Bose
Gerth Stølting Brodal
Andrej Brodnik
Sergio Cabello
Paz Carmi
Eric Colin de Verdiere
Mark de Berg
Erik Demaine
Vida Dujmovic
Sandor Fekete
Bob Fraser
Meng He
Matthew Katz
Rolf Klein
Jan Kratochvil
Stefan Langerman
Alejandro Lopez-Ortiz
Martin Milanič
Neža Mramor-Kosta
Ian Munro
Patrick Nicholson
Bengt J. Nilsson
Evanthia Papadopoulou
Tomaž Pisanski
Pedro Ramos
Borut Robič
Gunter Rote
Shakhar Smorodinsky
Bettina Speckmann
Marc van Kreveld
Uli Wagner
Primož Škraba
Borut Žalik

Additional Reviewers

Aistis Atminas
Stephane Durocher
Marko Grgurovič
Tatiana Romina Hartinger
Matevž Jekovec
Matjaž Konvalinka
Saeed Mehrabi
Debajyoti Mondal
Gelin Zhou

Table of Contents

Hierarchical Compositional Representations of Structure for Computer Vision and Robotics

Aleš Leonardis
University of Birmingham
School of Computer Science

Abstract

Modeling, learning, recognizing, and categorizing visual entities has been an area of intensive research in the vision and robotics communities for several decades. While successful partial solutions tailored for particular tasks and specific scenarios have appeared in recent years, more general solutions, which would be applicable to a variety of different tasks and would scale favorably with a large number of visual entities, are yet to be developed. Ultimately, the goal is to design and implement proper structures and mechanisms that would enable efficient learning, inference, and, when necessary, augmentation and modifications of the acquired visual knowledge in general scenarios. Recently, it has become increasingly clear that new approaches are needed to tackle these problems and there have been several indications that possible solutions should be sought in the framework of hierarchical architectures. Among various design choices related to hierarchies, compositional hierarchies show a great promise in terms of scalability, real-time performance, efficient structured on-line learning, shareability, and knowledge transfer. In this talk I will first present our work on compositional hierarchies related to visual representations of 2D and 3D object shapes and then conclude with some ideas towards generalizing the proposed approach to other visual entities and modalities.

Immersions of graphs and digraphs

Bojan Mohar
Simon Fraser University

Abstract

A graph G contains another graph H as an *immersion* if there is an injective mapping $\iota : V(H) \to V(G)$ and for each edge $uv \in E(H)$ there is a path P_{uv} in G joining vertices $\iota(u)$ and $\iota(v)$ such that the paths P_{uv} ($uv \in E(H)$) are pairwise edge-disjoint. If the paths are internally disjoint from $\iota(V(H))$, then we speak of a *strong immersion*. One can define (strong) immersions of digraphs in the same way.

Nash-Williams conjectured that graphs are well-quasi ordered for the relation of immersion containment. The conjecture was proved by Robertson and Seymour (Graph minors XXIII. Nash-Williams' immersion conjecture, J. Combinatorial Theory, Ser. B 100 (2010), 181–205) for weak immersions.

Recent interest in graph and digraph immersions resulted in a variety of new discoveries. The speaker will enlighten some of these achievements.

Computing Real Roots of Real Polynomials and its Application in Computational Geometry

Kurt Mehlhorn
Max-Planck-Institut für Informatik

Abstract

I also discuss recent advances in the computation of real roots of real polynomials. Near optimal solutions for the more general problem of isolating the complex roots of complex polynomials are known for quite some time (V. Pan, 2002). The new algorithms achieve the same time complexity for a sub-problem and are considerably simpler. I also discuss application to computational geometry, in particular, cylindrical algebraic decomposition. The talk is based on joint work with Michael Sagraloff.

A linear-time algorithm for the geodesic center of a simple polygon

Hee-Kap Ahn* Luis Barba[†,‡] Prosenjit Bose[†] Jean-Lou De Carufel[†] Matias Korman[§,¶]
Eunjin Oh*

Abstract

Given two points in a simple polygon P of n vertices, its geodesic distance is the length of the shortest path that connects them among all paths that stay within P. The geodesic center of P is the unique point in P that minimizes the largest geodesic distance to all other points of P. In 1989, Pollack, Sharir and Rote [Disc. & Comput. Geom. 89] showed an $O(n \log n)$-time algorithm that computes the geodesic center of P. Since then, a longstanding question has been whether this running time can be improved (explicitly posed by Mitchell [Handbook of Computational Geometry, 2000]). In this paper we affirmatively answer this question and present a linear time algorithm to solve this problem.

1 Introduction

Given a simple polygon P with n vertices and two points x, y in P, the *geodesic path* $\pi(x, y)$ is the shortest path contained in P connecting x with y. It is well-known that $\pi(x, y)$ is a polygonal chain whose vertices (other than its endpoints) are reflex vertices of P. The *geodesic distance* between x and y, denoted by $|\pi(x, y)|$, is the sum of the Euclidean lengths of each segment in $\pi(x, y)$. The *farthest neighbor* of $x \in P$ is a point whose geodesic distance to x is maximized. To ease the description, we assume that each vertex of P has a unique farthest neighbor. We can make this *general position* assumption using simulation of simplicity [4].

Let $F_P(x) : P \to \mathbb{R}$ be the function that maps a point $x \in P$ to the distance to its farthest neighbor (i.e., $F_P(x) = \max_{y \in P} |\pi(x, y)|$). A point $c_P \in P$ that minimizes $F_P(x)$ is called the *geodesic center* of P. Similarly, a point $s \in P$ that maximizes $F_P(x)$ (together with its farthest neighbor) is called a *geodesic diametral pair* and their distance is known as the *geodesic diameter*.

In 1983 Hershberger and Suri [7] presented a fast matrix search technique, one application of which is a linear-time algorithm for computing the diameter. Up to now, the best algorithm for computing the geodesic center is due to Pollack, Sharir, and Rote [11] and runs in $O(n \log n)$ time. Since then, it has been an open problem whether the geodesic center can be computed in linear time.

In this paper, we show how to compute the geodesic center of P in $O(n)$ time. Due to lack of space, proofs have been omitted in this document. A full version of the paper with all the omitted proofs can be found in [1].

2 Hourglasses and Funnels

Let $C \subseteq \partial P$ be a polygonal chain that starts at x and follows the boundary of P clockwise until reaching y. The *hourglass* of C, denoted by H_C, is the polygon contained in P bounded by C, $\pi(y, f(x))$, $\partial P(f(x), f(y))$ and $\pi(f(y), x)$. We call C and $\partial P(f(x), f(y))$ the *top* and *bottom* chains of H_C, respectively, while $\pi(y, f(x))$ and $\pi(f(y), x)$ are referred to as the *walls* of H_C. We say that the hourglass H_C is *open* if its walls are vertex disjoint. Note that open hourglasses are simple polygons and closed ones are weakly simple. We say C is a *transition chain* if $f(x) \neq f(y)$ and neither $f(x)$ nor $f(y)$ are interior vertices of C. In particular, if an edge xy of ∂P is a transition chain, we say that it is a *transition edge*.

Lemma 1 *[Rephrase of Lemma 3.1.3 of [2]] If C is a transition chain of ∂P, then the hourglass H_C is an open hourglass.*

Let $C = (p_0, \ldots, p_k)$ be a chain of ∂P and let v be a vertex of P not in C. The *funnel* of v to C, denoted by $S_v(C)$, is the simple polygon bounded by C, $\pi(p_k, v)$ and $\pi(v, p_0)$. See Lee and Preparata [8] or Guibas et al. [5] for more details on funnels.

A subset $R \subset P$ is *geodesically convex* if for every $x, y \in R$, the path $\pi(x, y)$ is contained in R. The (farthest) *Voronoi region* of a vertex v of P is the set of points $R(v) = \{x \in P : F_P(x) = |\pi(x, v)|\}$ (including boundary points).

*Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea. {heekap@postech.ac.kr, jin9082@postech.ac.kr}. Supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the Korea government (MSIP).

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

[‡]Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium. lbarbafl@ulb.ac.be

[§]National Institute of Informatics (NII), Tokyo, Japan. korman@nii.ac.jp

[¶]JST, ERATO, Kawarabayashi Large Graph Project.

Lemma 2 *Let v be a vertex of P and let C be a transition chain such $R(v) \cap \partial P \subseteq C$ and $v \notin C$. Then, $R(v)$ is contained in the funnel $S_v(C)$.*

3 Covering the boundary

In this section, we cover the boundary of P with sets of consecutive vertices that share the same farthest neighbor and edges of P whose endpoints have distinct farthest neighbors. Using a result from Hershberger and Suri [7], in $O(n)$ time we can compute the farthest neighbor of each vertex of P. Recall that the farthest neighbor of each vertex of P is always a convex vertex of P [3] and is unique by our general position assumption.

We mark the vertices of P that are farthest neighbors of at least one vertex of P. Let M denote the set of marked vertices of P. For each marked vertex v we create its funnel F_v as follows: let $u_1, \ldots, u_{k-1}$ be the vertices of P such that $v = f(u_i)$ and assume that they appear in this order when traversing ∂P clockwise. Let u_0 and u_k be the neighbors of u_1 and u_{k-1} other than u_2 and u_{k-2}, respectively. Note that both u_0u_1 and $u_{k-1}u_k$ are transition edges of P. The funnel F_v is defined as the funnel $S_v(C_v)$, where $C_v = (u_0, \ldots, u_k)$.

For each transition edge of ∂P, we also consider its associated hourglass. Let $\mathcal{C}_P$ be the union of all such hourglasses and funnels. We call $\mathcal{C}_P$ the *covering* of P into funnels and hourglasses.

Lemma 3 *For any simple polygon P its covering $\mathcal{C}_P$ is a collection of simple polygons whose overall complexity is $O(n)$. Moreover, we can explicitly compute $\mathcal{C}_P$ in linear time, and it holds that $\bigcup_{U \in \mathcal{C}_P} U = P$.*

4 Covering the polygon with apexed triangles

An *apexed triangle* $\triangle = (a, b, c)$ with *apex* a is a triangle contained in P with an associated distance function $g_\triangle(x)$, called the *apex function* of $\triangle$, such that (1) a is a vertex of P, (2) $b, c \in \partial P$, and (3) there is a vertex w of P, called the *definer* of $\triangle$, such that

$$g_\triangle(x) = \begin{cases} -\infty & \text{if } x \notin \triangle \\ |xa| + |\pi(a,w)| = |\pi(x,w)| & \text{if } x \in \triangle \end{cases}$$

In this section, we show how to find a set of $O(n)$ apexed triangles of P such that the upper envelope of their apex functions coincides with $F_P(x)$. Since the decomposition of P into hourglasses and funnels covers P, we look at each element of $\mathcal{C}_P$ independently. We consider both cases independently.

Let ab be a transition edge of P such that b is the clockwise neighbor of a along ∂P. Let B_{ab} denote the bottom chain of H_{ab} after removing its endpoints. For every vertex v that lies between $f(a)$ and $f(b)$ in the bottom chain of H_{ab}, we know that there cannot be a vertex u of P such that $f(u) = v$. As proved by Aronov et al. [2, Corollary 2.7.4], if there is a point x on ∂P whose farthest neighbor is v, then x must lie on the open segment (a, b). In other words, for any vertex v of B_{ab} such that $R(v) \neq \emptyset$, then $R(v) \cap \partial P \subset ab$. In fact, not only this Voronoi region is inside H_{ab} when restricted to the boundary of P, but also $R(v) \subset H_{ab}$.

The next result follows trivially from Lemma 2.

Corollary 4 *Let v be a vertex of B_{ab}. If $R(v) \neq \emptyset$, then $R(v) \subset H_{ab}$.*

Our objective is to compute $O(|H_{ab}|)$ apexed triangles that cover H_{ab}, each with its distance function, such that the upper envelope of these apex functions coincides with $F_P(x)$ restricted to H_{ab} where it "matters". The same approach was already used by Pollack et al. in [11, Section 3]. Given a segment contained in the interior of P, they show how to compute a linear number of apexed triangles such that $F_P(x)$ coincides with the upper envelope of the corresponding apex functions in the given segment.

Let T_a and T_b be the shortest path trees in H_{ab} from a and b, respectively. For each vertex v between $f(a)$ and $f(b)$, let v_a and v_b be the neighbors of v in the paths $\pi(v, a)$ and $\pi(v, b)$, respectively. We say that a vertex v is *visible* from ab if $v_a \neq v_b$. For each visible vertex v, we obtain a triangle $\triangle_v$.

We further split $\triangle_v$ into a series of triangles with apex at v as follows: Let u be a child of v in either T_a or T_b. As noted by Pollack et al., v can be of three types, either (1) u is not visible from ab (and is hence a child of v in both T_a and T_b); or (2) u is visible from ab, is a child of v only in T_b, and v_bvu is a left turn; or (3) u is visible from ab, is a child of v only in T_a, and v_avu is a right turn.

Let $u_1, \ldots, u_{k-1}$ be the children of v of type (2) sorted in clockwise order around v. Let $c(v)$ be the maximum distance from v to any invisible vertex in the subtrees of T_a and T_b rooted at v; if no such vertex exists, then $c(v) = 0$. Define a function $d_l(v)$ on each vertex v of H_{ab} in a recursive fashion as follows: If v is invisible from ab, then $d_l(v) = c(v)$. Otherwise, let $d_l(v)$ be the maximum of $c(v)$ and $\max\{d_l(u_i) + |u_iv| : u_i \text{ is a child of } v \text{ of type (2)}\}$. Similarly we define a symmetric function $d_r(v)$ using the children of type (3) of v.

For each $1 \leq i \leq k-1$, extend the segment u_iv past v until it intersects ab at a point s_i. Let s_0 and s_k be the intersections of the extensions of vv_a and vv_b with the segment ab. We define then k triangles contained in $\triangle_v$ as follows. For each $0 \leq i \leq k-1$, consider the triangle $\triangle(s_i, v, s_{i+1})$ whose associated apexed (left) function is

$$f_i(x) = |xv| + \max_{j>i}\{c(v), |vu_j| + d_l(u_j)\}.$$

In a symmetric manner, we define a set of apexed triangles induced by the type (3) children of v and their respective apexed (right) functions.

Let $g_1, \ldots, g_r$ and $\triangle_1, \ldots, \triangle_r$ respectively be an enumeration of all the generated apex functions and triangles such that g_i is defined in the triangle $\triangle_i$. Note that for each $1 \leq i \leq r$, the triangle $\triangle_i$ has two vertices on the segment ab and a third vertex, say a_i, called its *apex* such that for each $x \in \triangle_i$, $g_i(x) = |\pi(x, w_i)|$ for some vertex w_i of H_{ab}. We refer to w_i as the *definer* of $\triangle_i$. Intuitively, $\triangle_i$ defines a portion of the geodesic distance function from w_i in a constant complexity region.

Lemma 5 *Given a transition edge ab of P, we can compute a set $\mathcal{A}_{ab}$ of $O(|H_{ab}|)$ apexed triangles in $O(|H_{ab}|)$ time with the property that for any point $p \in P$ such that $f(p) \in B_{ab}$, there is an apexed function g such that $g(p) = F_P(p)$.*

Inside the funnels of marked vertices

For each marked vertex $v \in M$ we have constructed the funnel $S_v(C_v)$ such that v is the farthest neighbor of all vertices of C_v other than its endpoints. We call $C_v = (u_0, \ldots, u_k)$ the *main chain* of $S_v(C_v)$ while $\pi(u_k, v)$ and $\pi(v, u_0)$ are referred to as the *walls* of the funnel.

Lemma 6 *Let $x \in P$ such that $f(x) = v$ for some marked vertex $v \in M$. Then, it holds that $x \in S_v(C_v)$.*

As with the hourglass case, we need to split a funnel into $O(|S_v(C_v)|)$ apexed triangles that encode the distance function from v. To this end, we compute the shortest path tree T_v of v in $S_v(C_v)$ in $O(|S_v(C_v)|)$ time [6]. We consider the tree T_v to be rooted at v and assume that for each node u of this tree we have stored the geodesic distance $|\pi(u, v)|$.

Start an Eulerian tour from v walking in a clockwise order of the edges. Let Let w_1 be the first leaf of T_v found, and let w_2 and w_3 be the next two vertices visited in the traversal. Two cases arise:

Case 1: w_1, w_2, w_3 makes a right turn. We define s as the first point hit by the ray apexed at w_2 that shoots in the direction opposite to w_3. In this case, we construct the apexed triangle $\triangle(w_2, w_1, s)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. We modify tree T_v by removing the edge w_1w_2 and replacing the edge w_3w_2 by the edge w_3s.

Case 2: w_1, w_2, w_3 makes a left turn and w_1 and w_3 are adjacent, then if w_1 and w_3 lie on the same edge of ∂P, we construct an apexed triangle $\triangle(w_2, w_1, w_3)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. Otherwise, let s be the first point of the boundary of $S_v(C_v)$ hit by the ray shooting from w_3 in the direction opposite to w_2. We construct an apexed triangle $\triangle(w_2, w_1, s)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. We modify the tree T_v by removing the edge w_1w_2 and adding the edge w_3s.

Lemma 7 *The above procedure runs in $O(|S_v(C_v)|)$ time and computes $O(|S_v(C_v)|)$ interior disjoint apexed triangles such that their union covers $S_v(C_v)$. Moreover, for each point $x \in R(v)$ there is apex function $g(x)$ such that $g(x) = F_P(x)$.*

5 Prune and search

With the tools introduced in the previous sections, we can proceed to give the prune and search algorithm to compute the geodesic center. The idea of the algorithm is to partition P into $O(1)$ cells, determine on which cell of P the center lies and recurse on that cell as a new subproblem with smaller complexity.

Let τ be the set all apexed triangles computed in previous sections. Lemmas 5 and 7 directly provide a $O(n)$ bound on the complexity of τ.

Let $\phi(x)$ be the upper envelope of the apex functions of every triangle in τ (i.e., $\phi(x) = \max\{g_i(x) : g_i(x) \in \tau, x \in \triangle_i\}$). Lemmas 5 and 7 imply that the $O(n)$ apexed triangles of τ not only cover P, but their apex functions suffice to reconstruct the function $F_P(x)$. That is, for each $p \in P$, $\phi(p) = F_P(p)$.

Given a chord C of P, a *half-polygon* of P is one of the two simple polygons in which C splits P. A *4-cell* of P is a simple polygon obtained as the intersection of at most four half-polygons. Because a 4-cell is the intersection of geodesically convex sets, it is also geodesically convex.

Let R be a 4-cell of P and let τ_R be the set of apexed triangles of τ that intersect R. Let $m_R = \max\{|R|, |\tau_R|\}$. Recall that, by construction of the apexed triangles, for each triangle of τ_R at least one and at most two of its boundary segments is a chord of P. Let $\mathcal{C}$ be the set containing all chords that belong to the boundary of a triangle of τ_R. Therefore, $|\tau_R| \leq |\mathcal{C}| \leq 2|\tau_R|$.

To construct an ε-net of $\mathcal{C}$, we need some definitions (for more information on ε-nets refer to [9]). Let φ be the set of all open 4-cells of P. For each $t \in \varphi$, let $\mathcal{C}_t = \{C \in \mathcal{C} : C \cap t \neq \emptyset\}$ be the set of chords of $\mathcal{C}$ induced by t. Finally, let $\varphi_{\mathcal{C}} = \{\mathcal{C}_t : t \in \varphi\}$ be the family of subsets of $\mathcal{C}$ induced by φ.

Let $\varepsilon > 0$ (the exact value of ε will be specified later). Consider the range space $(\mathcal{C}, \varphi_{\mathcal{C}})$ defined by $\mathcal{C}$ and $\varphi_{\mathcal{C}}$. Because the VC-dimension of this range space is finite, we can compute an ε-net N of $(\mathcal{C}, \varphi_{\mathcal{C}})$ in $O(n)$ time [9]. The size of N is $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) = O(1)$ and its main property is that any 4-cell that does not intersect a chord of N will intersect at most $\varepsilon|\mathcal{C}|$ chords of $\mathcal{C}$.

Observe that N partitions R into $O(1)$ subpolygons (not necessarily 4-cells). We further refine this partition by performing a 4-cell decomposition. That is, we shoot vertical rays up and down from each endpoint of N, and from the intersection point of any two segments of N. Overall, this partitions R into $O(1)$ 4-cells such that each either (i) is a convex polygon contained in P of at most four vertices, or otherwise (ii) contains some chain of ∂P. Since $|N| = O(1)$, the whole decomposition can be computed in $O(m_R)$ time.

In order to determine which 4-cell contains the geodesic center of P, we extend each edge of a 4-cell to a chord C. We then use the chord-oracle from Pollack et al. [11, Section 3] to decide which side of C contains c_P. Since this oracle runs in time proportional to the number of functions defined on C, we can decide in total $O(m_R)$ time on which side of C the geodesic center of P lies. Since our decomposition into 4-cells has constant complexity, $O(1)$ calls are needed to determine the 4-cell R' containing the geodesic center of P. Since N is a ε-net, we know that at most $\varepsilon|\mathcal{C}|$ chords of $\mathcal{C}$ will intersect R'.

Using a similar argument, we can show that the complexity of R' also decreases: each vertex of R' must be in at least one apexed triangle of τ_R. By construction, each apexed triangle can cover at most three vertices. By the pigeonhole principle we conclude that R' can have at most $6\varepsilon m_R$ vertices. Thus, if we choose $\varepsilon = 1/12$, we guarantee that both the size of the 4-cell R' and the number of apexed triangles in $\tau_{R'}$ are at most $m_R/2$.

We proceed recursively on R', and obtain that after $O(\log m_R)$ iterations, we reduce the size of either τ_R or R' to constant. In the former case, the minimum of $F_P(x)$ can be found by explicitly constructing function ϕ in $O(1)$ time. In the latter case, we triangulate R' and apply the chord-oracle to determine which triangle will contain c_P.

Thus, in order to complete the algorithm it remains to show how to find the geodesic center of P for the case in which R' is a triangle. Recall that $\phi(x)$ denotes the upper envelope of the apex functions of the triangles in τ, and the geodesic center is the point that minimizes ϕ. The key observation is that, as it happened with chords, the function $\phi(x)$ restricted to R' is convex. Following the approach of Meggido [10], we transform our problem into an equivalent optimization problem in $\mathbb{R}^3$ (by *lifting* the apexed functions).

We use a prune and search approach similar to the previous one: pair the functions arbitrarily, and consider the set of $m/2$ bisectors defined by these pairs. For some constant r, compute a $1/r$-cutting, and determine in which of the regions contains the minimum. At least $(r-1)m/2r$ separating planes do not intersect this constant size region, and for each of them we can discard one of the constraints. The main difficulty is that apex functions are only defined in a triangular domain. In particular, the bisector between two such functions is not properly defined. Details are omitted.

The following theorem summarizes the results presented in this paper.

Theorem 8 *We can compute the geodesic center of any simple polygon P of n vertices in $O(n)$ time.*

References

[1] H.-K. Ahn, L. Barba, P. Bose, J.-L. D. Carufel, M. Korman, and E. Oh. A linear-time algorithm for the geodesic center of a simple polygon. *CoRR*, abs/1501.00561, 2015.

[2] B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(1):217–255, 1993.

[3] T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, 1985.

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

[5] 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.

[6] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In *Proc. of STOC*, pages 50–63, 1987.

[7] J. Hershberger and S. Suri. Matrix searching with the shortest path metric. In *Proc. of STOC*, pages 485–494, 1993.

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

[9] J. Matoušek. Construction of epsilon nets. In *Proc. of SoCG*, pages 1–10, New York, 1989.

[10] N. Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(1):605–610, 1989.

[11] R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(1):611–626, 1989.

Computing the s-kernel of Orthogonal Polygons*

Leonidas Palios†

Abstract

Two points p, q of an orthogonal polygon P are s-visible from one another if there exists a staircase path (i.e., an x- and y-monotone chain of horizontal and vertical line segments) from p to q that lies in P. The s-kernel of P is the (possibly empty) set of points of P from which all points of P are s-visible.

We are interested in the problem of computing the s-kernel of a given orthogonal polygon (on n vertices) possibly with holes. The problem has been considered by Gewali [1] who described an $O(n)$-time algorithm for orthogonal polygons without holes and an $O(n^2)$-time algorithm for orthogonal polygons with holes. The problem is a special case of the problem considered by Schuierer and Wood [3], whose work implies an $O(n)$-time algorithm for orthogonal polygons without holes and an $O(n \log n + h^2)$-time algorithm for orthogonal polygons with $h \geq 1$ holes.

In this paper, we give a simple output-sensitive algorithm for the problem. For an n-vertex orthogonal polygon P that has h holes, our algorithm runs in $O(n + h \log h + k)$ time where $k = O(1 + h^2)$ is the number of connected components of the s-kernel of P. Moreover, a modified version of our algorithm enables us to compute the number k of connected components of the s-kernel in $O(n + h \log h)$ time.

1 Introduction

A polygon is *orthogonal* if its edges are either horizontal or vertical; an edge e of such a polygon is a N-edge (S-edge, E-edge, and W-edge, resp.) if the outward-pointing normal vector to e is directed towards the North (South, East, and West, resp.); see Figure 1(left). Of particular importance are the *dents*, i.e., edges whose endpoints are reflex vertices of the polygon, characterized as N-dents, S-dents, E-dents, and W-dents (see Figure 1(left)); the dents are a measure of non-convexity of an orthogonal polygon.

A set of points is *x-monotone* (*y-monotone*, resp.) if its intersection with any line perpendicular to the x-axis (y-axis, resp.) is a connected set. A *staircase path* is a chain of horizontal and vertical segments that is both x- and y-monotone. Then, two points p, q of an orthogonal polygon P are *s-visible* from one another if there exists a staircase path from p to q that lies in P (see Figure 1(left)). The *s-kernel* of P is the (possibly empty) set of points of P from which all points of P are s-visible. An orthogonal polygon is an *s-star* if it has non-empty s-kernel. Note that an s-star may have holes (see Figure 1(right)).

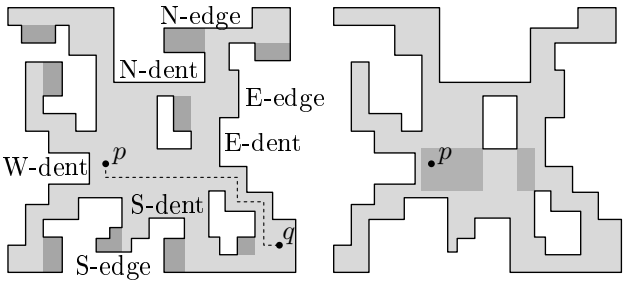


Figure 1: (left) Illustration of the main definitions (the portions of the polygon not s-visible from p are shown dark); (right) the s-star of p with its s-kernel shown darker.

Visibility problems are closely related to reachability and to covering problems. The s-kernel of a polygon is the set of points from which all other points of the polygon can be reached by means of x- and y-monotone paths. So, if a robot restricted to move parallel to the coordinate axes "guards" a point p in an orthogonal polygon provided that it can get to p along a monotone path, then the polygons that can be "guarded" are those with non-empty s-kernel. Additionally, because the s-stars may be highly non-convex (see Figure 1(right)), a minimum cover of an orthogonal polygon using s-stars (see [2] for an algorithm) is expected to involve a smaller number of pieces compared to other minimum covers.

Gewali [1] has considered the problem of computing the s-kernel of an orthogonal polygon; he described an $O(n)$-time algorithm for an n-vertex orthogonal polygon without holes and an $O(n^2)$-time algorithm for orthogonal polygons with holes. Gewali also showed that the latter algorithm is worst-case optimal since the s-kernel of an orthogonal polygon with holes may be of $\Theta(n^2)$ size, and used it to give an $O(n \log n)$-time algorithm for recognizing whether an orthogonal

* This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS UOA (MIS 375891) - Investing in knowledge society through the European Social Fund.

† Department of Computer Science and Engineering, University of Ioannina, Greece.

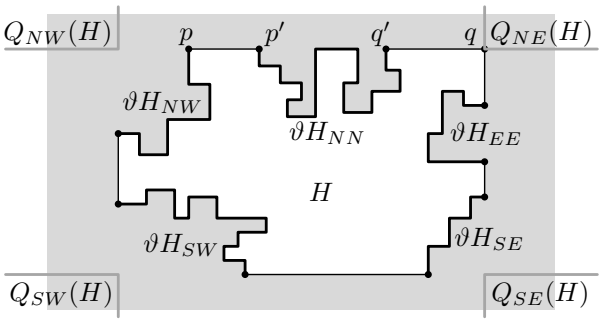


Figure 2: Illustration of the notation for a hole H (the subchain ϑH_{NE} is point q; no $\vartheta H_{WW}, \vartheta H_{SS}$ exist).

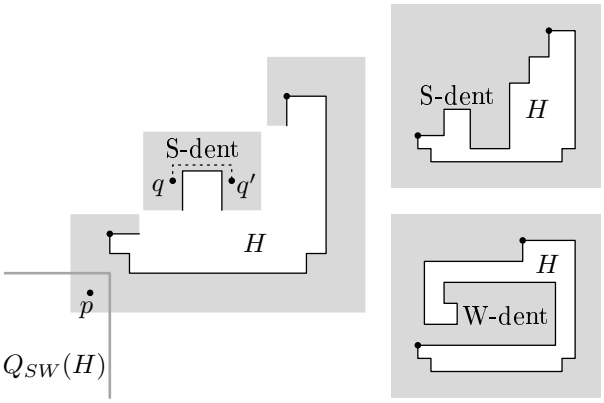


Figure 3: For Lemma1(ii): if ϑH_{NW} contains a S-dent or an W-dent, then no point of the quadrant $Q_{SW}(H)$ belongs to the s-kernel of P.

polygon with holes is an s-star. Schuierer and Wood [3] studied the notion of $\mathcal{O}$-visibility, that is, visibility along a set $\mathcal{O}$ of orientations and gave an $O(n \log |\mathcal{O}|)$-time and an $O(n(\log |\mathcal{O}| + \log n) + h(|\mathcal{O}| + h))$-time algorithm for the computation of the $\mathcal{O}$-kernel of a simple polygon and of a polygon with h holes, respectively. Their algorithms imply $O(n)$-time and $O(n \log n + h^2)$-time algorithms for the s-kernel of a simple orthogonal polygon and of an orthogonal polygon with $h \geq 1$ holes, respectively.

In this paper, we present a simple output-sensitive $O(n + h \log h + k)$-time and $O(n)$-space algorithm for computing the s-kernel of an orthogonal polygon having n vertices, $h \geq 0$ holes, and an s-kernel consisting of $k = O(1 + h^2)$ connected components. The algorithm also enables us to count the number k of connected components of the s-kernel of such a polygon in $O(n + h \log h)$ time using $O(n)$ space (i.e., without computing the s-kernel), and thus we can determine if an orthogonal polygon is an s-star in the same time and space complexity.

2 Theoretical Framework

For an edge e of an orthogonal polygon P, let D_e be a small enough disk centered at the midpoint of e; we define the *in-halfplane* of e as the *closed* halfplane that is defined by the line supporting e and contains $D_e \cap P$. An orthogonal polygon is *orthogonally convex* if it is both x-monotone and y-monotone. It easily follows that a polygon is orthogonally convex if and only if it has no dents.

Notation: Let D be an orthogonal polygon or a hole in an orthogonal polygon. Then, we define:

ϑD**:** the boundary of D;

$BBox(D)$**:** the smallest axis-parallel rectangle containing D.

Additionally, for a hole H, we have:

$S_{=}(H)$**:** the smallest *open* horizontal strip containing the interior of H;

$S_{||}(H)$**:** the smallest *open* vertical strip containing the interior of H;

$Q_{NW}(H)$**:** the *closed* quadrant that is the complement of the union of the interiors of the inhalfplanes of the top and left edges of the rectangle $BBox(H)$ (see Figure 2) — similarly, we define $Q_{NE}(H)$, $Q_{SW}(H)$, and $Q_{SE}(H)$;

ϑH_{NW}**:** the part of the boundary of H in counterclockwise direction from the leftmost among the points of H with maximum y-coordinate to the topmost among the points of H with minimum x-coordinate (see Figure 2) — similarly, we define ϑH_{NE}, ϑH_{SW}, and ϑH_{SE};

ϑH_{NN}**:** let p, q be the leftmost and rightmost, resp., vertices of H with maximum y-coordinate; if p, q are adjacent in H then no ϑH_{NN} exists; otherwise, if p' (q', resp.) is the other endpoint of the horizontal edge incident on p (q, resp.), ϑH_{NN} is the part of the boundary of H connecting p' and q' after the edges pp' and qq' have been removed (see Figure 2) — similarly, we define ϑH_{WW}, ϑH_{SS}, and ϑH_{EE}.

The following lemma provides important properties of the s-kernel of orthogonal polygons with holes.

Lemma 1 *Let H be a hole of an orthogonal polygon P. Then:*

(i) No point of the strips $S_{=}(H)$ and $S_{||}(H)$ belongs to the s-kernel of P.

(ii) If ϑH_{NW} is not a single point, then no point of the quadrant $Q_{SE}(H)$ belongs to the s-kernel of P. Moreover:
if ϑH_{NW} contains a S-dent or an W-dent, then no point of the quadrant $Q_{SW}(H)$ belongs to the s-kernel of P;

if ϑH_{NW} contains a N-dent or an E-dent, then no point of the quadrant $Q_{NE}(H)$ belongs to the s-kernel of P;
if ϑH_{NW} contains a N-dent or an W-dent, then no point of the quadrant $Q_{NW}(H)$ belongs to the s-kernel of P.
Similar results hold for the boundary subchains ϑH_{NE}, ϑH_{SW}, and ϑH_{SE}.

(iii) *If the boundary of H contains a subchain ϑH_{NN}, then no point of the quadrants $Q_{SW}(H) \cup Q_{SE}(H)$ belongs to the s-kernel of P. Moreover:*
if ϑH_{NN} contains a N-dent or an E-dent, then no point of the quadrant $Q_{NE}(H)$ belongs to the s-kernel of P;
if ϑH_{NN} contains a N-dent or an W-dent, then no point of the quadrant $Q_{NW}(H)$ belongs to the s-kernel of P.
Similar results hold for the boundary subchains ϑH_{WW}, ϑH_{SS}, and ϑH_{EE}.

The above lemma implies that for a hole H of the given orthogonal polygon P, points of the s-kernel of P belong to all, some, or none of the four quadrants $Q_{NW}(H)$, $Q_{NE}(H)$, $Q_{SW}(H)$, and $Q_{SE}(H)$.

The algorithm of Gewali [1] computes the s-kernel of a simple orthogonal polygon P by intersecting P with the in-halfplanes of the lowermost N-dent, the rightmost W-dent, the topmost S-dent, and the leftmost E-dent. This implies the following result.

Lemma 2 *Let P be an orthogonal polygon without holes that has n vertices. The s-kernel of P is an orthogonally convex orthogonal polygon of $O(n)$ size.*

3 Computing the s-Kernel

Let P be an orthogonal polygon. As in [3], we compute the s-kernel A of P ignoring the holes in P, which we intersect with the *external* s-kernel of each of P's holes. In order to compute the external s-kernel of a hole, we process it determining its horizontal and vertical strips and quadrants not containing points of the s-kernel of P (as described in Lemma 1); then, the intersection of their external s-kernels is the complement of the union of the collected strips and quadrants. Note that in order to get a fast algorithm, we do not intersect the complements of the horizontal and vertical strips; instead, we process each horizontal strip I containing points of the s-kernel and determine the s-kernel components (if any) in I taking into account the vertical strips. A detailed description of the algorithm is given in Algorithm s-KERNEL below.

The correctness of Algorithm s-KERNEL follows from Lemma 1 and the fact that the s-kernel of P is indeed the intersection of polygon A (see Step 1) with the complement of the union of the collected strips and quadrants from the holes of P.

Algorithm s-KERNEL(P)
Input : an orthogonal polygon P possibly with holes
Output: the s-kernel of P

---

1. compute the s-kernel A of the orthogonal polygon bounded only by P's outer boundary component;
   if P has no holes
   then return A as the s-kernel of P;
   exit;
   let $x_{min}, x_{max}, y_{min}, y_{max}$ be the extreme values of x- and y-coordinates of the bounding rectangle $BBox(A)$ of A;

2. process the holes of P to determine the (open) strips and (closed) quadrants that do not contain points of the s-kernel of P (see Lemma 1);
   if all 4 quadrants $Q_{NW}(H)$, $Q_{NE}(H)$, $Q_{SW}(H)$, $Q_{SE}(H)$ of a hole H do not contain points of the s-kernel of P
   then print("The s-kernel of P is empty.");
   exit;
   let $\mathcal{C}_{||}$ ($\mathcal{C}_{=}$, $\mathcal{C}_Q$, resp.) be the set of vertical strips (horizontal strips, quadrants, resp.) not containing points of the s-kernel of P;

3. {*process the strips in $\mathcal{C}_{||}$ and $\mathcal{C}_{=}$*}
   compute the union of the vertical strips in $\mathcal{C}_{||}$, clip it about the range $[x_{min}, x_{max}]$, and store it in an x-ordered array $M_{||}$ of alternating *closed* "in"-strips (containing points of the s-kernel) and *open* "out"-strips (not containing points of the s-kernel);
   work similarly for the horizontal strips in $\mathcal{C}_{=}$ using the range $[y_{min}, y_{max}]$;

4. {*process the quadrants in $\mathcal{C}_Q$*}
   compute the union U_Q of all the quadrants in $\mathcal{C}_Q$, and clip its complement about the boundary of the polygon A computed in Step 1;

5. **for** each polygon B_j in the clipped complement of U_Q in y-order **do**
   for each horizontal "in"-strip I intersecting B_j in y-order **do**
   compute the boundary $\vartheta B_j(I) = \vartheta B_j \cap I$;
   locate a leftmost point of $\vartheta B_j(I)$ in the vertical strips array $M_{||}$;
   walk on $\vartheta B_j(I)$ and in $M_{||}$ until a rightmost point of $\vartheta B_j(I)$ is found, printing each polygon (if any) contributed by $B_j \cap I$ and each "in"-strip of $M_{||}$;

(Note that the clipped complement of the union U_Q at the completion of Step 4 does not contain its entire boundary; it contains the edges that resulted from the clipping about A but it does not contain the edges that resulted from the quadrants in $\mathcal{C}_Q$.)

Time and Space Complexity. Let n and h be the number of vertices and holes of the input orthogonal polygon P. We prove the following lemma.

Lemma 3 *(i) Each halfline bounding a quadrant in $\mathcal{C}_Q$ contributes at most one edge to the polygons forming the complement of the union U_Q of all the quadrants in $\mathcal{C}_Q$.*

(ii) The clipped complement of U_Q computed upon completion of Step 4 consists of $O(h)$ horizontally and vertically separated (i.e., no horizontal or vertical line intersects two of them) orthogonally convex orthogonal polygons of $O(n)$ total size.

The computation of the s-kernel in Step 1 takes $O(n)$ time [1] and so does the entire Step 1. Step 2 takes $O(n)$ time as well by traversing the boundary of each of the h holes of P and applying Lemma 1. The processing of the h vertical strips in $\mathcal{C}_{||}$ in Step 3 can be completed in $O(h \log h)$ time by sorting them by non-decreasing left side and then processing them from left to right; similarly, the processing of the horizontal strips in $\mathcal{C}_{=}$ takes $O(h \log h)$ time. In Step 4, we sort the quadrants in y-order in $O(h \log h)$ time and compute the right-bounding line of the union of quadrants $Q_{NW}(H_i)$ and $Q_{SW}(H_{i'})$ in $\mathcal{C}_Q$ and the left-bounding line of the union of quadrants $Q_{NE}(H_i)$ and $Q_{SE}(H_{i'})$ in $O(h)$ time. The complement of these unions is clipped about polygon A and by traversing their boundaries from top to bottom we compute the clipped complement of U_Q in $O(n)$ time. In total, Step 4 takes $O(n + h \log h)$ time. For Step 5, let t_j be the number of horizontal "in"-strips intersecting polygon B_j. Because the polygons in the clipped complement of U_Q are horizontally separated (Lemma 3(ii)), then any other polygon may be intersected only by the topmost or bottommost of these t_j "in"-strips. Then, the number of pairs of polygons and "in"-strips considered is $\sum_j t_j = \sum_j 2 + \sum_j (t_j - 2) = O(h)$ since the numbers of polygons and "in"-strips are both $O(h)$. Thus, if the s-kernel of P has k conn. components, Step 5 takes $O(n + h \log h + k)$ time by using binary search for locating leftmost points. Therefore:

Theorem 4 *Let P be an orthogonal polygon having n vertices and $h = O(n)$ holes. Algorithm s-KERNEL computes the s-kernel of P in $O(n + h \log h + k)$ time using $O(n)$ space where k is the number of connected components of the s-kernel of P.*

4 Number of Components of the s-Kernel

Algorithm s-KERNEL can be modified to help us compute the number $k = O(1 + h^2)$ of connected components of the s-kernel of a given orthogonal polygon P; it suffices to modify Step 1 so that if P has no holes it returns 0 if A is empty and 1 otherwise, Step 2 to return 0 if the s-kernel is found empty, and Step 5 as follows: for each polygon B_j and each horizontal "in"-strip I intersecting B_j, we compute a leftmost point a and a rightmost point z of the boundary of B_j in I, and locate them in the vertical strips array $M_{||}$; then, depending on the indices of the strips to which a, z belong and whether they are "in"- or "out"-strips, we compute the number $\kappa(B_j, I)$ of "in"-strips (if any) between (and including) the strips of a and of z. The total number of components of the s-kernel of P is the sum of all the computed $\kappa(B_j, I)$.

The correctness of the modified algorithm follows immediately from the fact that for each polygon B_j and each horizontal "in"-strip I, each"in"-strip between (and including) a and z contributes a separate component to the s-kernel of P. The complexity analysis of Step 5 of Algorithm s-KERNEL and the fact that $\kappa(B_j, I)$ can be computed in constant time imply that the modified Step 5 takes $O(n + h \log h)$ time.

The modified algorithm readily implies an algorithm to recognize whether a polygon P is an s-star (i.e., its s-kernel consists of at least 1 component) or not. A simpler version that does not compute the number k of components simply checks in Step 5 whether a, z fall in the same vertical "out"-strip of $M_{||}$; if they don't, then there exists a point in $B_j \cap I$ belonging to the s-kernel of P and hence P is an s-star (the algorithm can be augmented to return such a point as a certificate of its decision). If the above condition for a, z does not hold for all polygons B_j and "in"-strips I, then P is not an s-star. In summary:

Theorem 5 *Let P be an orthogonal polygon having n vertices and $h = O(n)$ holes. The described modified algorithm computes the number of conn. components of the s-kernel of P in $O(n + h \log h)$ time using $O(n)$ space. Moreover, it can be decided whether P is an s-star in the same time and space complexity.*

5 Open Problems

We leave as open problems the study of the complexity status of the s-star recognition problem (i.e., can there be an algorithm running in $o(n + h \log h)$ time or is there an $\Omega(n + h \log h)$ lower bound?) and of extensions of the problem to 3-dimensional space.

References

[1] L.P. Gewali. Recognizing s-Star Polygons. *Pattern Recognition* 28(7), 1019-1032, 1995

[2] R. Motwani, A. Raghunathan, and H. Saran. Covering Orthogonal Polygons with Star Polygons: the Perfect Graph Approach. *J. Comput. Systems Science* 40, 19-48, 1990

[3] S. Schuierer and D. Wood. Generalized Kernels of Polygons with Holes. *Proc. 5th Canadian Conf. on Computational Geometry*, 222-227, 1993

Optimizing an oriented convex hull with two directions

Carlos Alegría-Galicia * David Orden † Carlos Seara ‡ Jorge Urrutia §

Abstract

Given a set P of n points in the plane in general position, we generalize the rectilinear convex hull of P, $\mathcal{RH}(P)$, to the $\mathcal{O}^2_\beta$-convex hull of P, denoted by $\mathcal{O}^2_\beta\mathcal{H}(P)$, where the directions of two oriented lines, used as coordinate axes, form an angle $\beta \in [0, \pi]$. We show: (i) How this hull can be computed and maintained while β changes in $[0, \pi]$, and (ii) How to determine the angle β for which $\mathcal{O}^2_\beta\mathcal{H}(P)$ maximizes its area or minimizes its perimeter. Our algorithms run in optimal $\Theta(n \log n)$ time and $O(n)$ space.

1 Introduction

All the point sets P considered in this paper will be assumed in general position and such that no two elements of P lie on a horizontal line. Let $\mathcal{O}^k$ be a set of k lines in the plane through a common point. A region R in the plane is called $\mathcal{O}^k$*-convex* if its intersection with any line parallel to one in $\mathcal{O}^k$ is either empty or connected, see [4, 7].

Ottmann et al. [5] consider $k = 2$ with horizontal and vertical lines, showing how to compute the so-called rectilinear convex hull of P, denoted by $\mathcal{RH}(P)$, in optimal $\Theta(n \log n)$ time and $O(n)$ space. Rotating the set of two lines makes $\mathcal{RH}(P)$ change and the rotation for which $\mathcal{RH}(P)$ has minimum area was obtained in [1], in optimal $\Theta(n \log n)$ time and $O(n)$ space. See [2] for a generalization.

Here we also consider the case $k = 2$, with a set $\mathcal{O}^2$ composed of a horizontal line (oriented from left to right) and a second line (oriented from bottom to top) forming an angle β with the horizontal, see Figure 1 (left). Hence, we may denote $\mathcal{O}^2$ as $\mathcal{O}^2_\beta$.

Following Ottmann [5], we define the $\mathcal{O}^2_\beta$-convex hull of a point set P as the intersection of all the connected supersets of P which are $\mathcal{O}^2_\beta$-convex, see Figure 1 (right). The $\mathcal{O}^2_\beta$-convex hull of a point set P will be denoted as $\mathcal{O}^2_\beta\mathcal{H}(P)$. In this paper we show algorithms for: (i) Computing and maintaining $\mathcal{O}^2_\beta\mathcal{H}(P)$ while β changes in $[0, \pi]$, and (ii) finding an angle $\beta \in [0, \pi]$ such that the area of $\mathcal{O}^2_\beta\mathcal{H}(P)$ is maximized or the (non-zero) perimeter of $\mathcal{O}^2_\beta\mathcal{H}(P)$ is minimized. Our algorithms run in $\Theta(n \log n)$ time and $O(n)$ space.

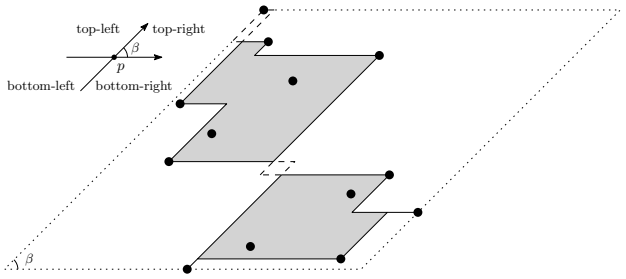


Figure 1: left: Example of $\mathcal{O}^2$. right: Example of $\mathcal{O}^2_\beta\mathcal{H}(P)$.

Let $\mathcal{D} = \{0, \beta, \pi, \pi + \beta\}$. Consider two consecutive elements o_1 and o_2 in $\mathcal{D}$, in the counterclockwise order, and a point p on the plane. The *stabbing* $\mathcal{O}^2_\beta$*-wedge associated to* o_1, o_2 with apex p is the open region bounded between two rays emanating from p with orientations o_1 and o_2, respectively. Note that every point p in the plane is the apex of four stabbing $\mathcal{O}^2_\beta$-wedges; top-left, top-right, bottom left, and bottom-right. See Figure 1 (left).

Proposition 1 ([2]) *Let $\mathcal{W}$ be the set of all stabbing $\mathcal{O}^2_\beta$-wedges of the plane containing no elements of P. The $\mathcal{O}^2_\beta$-convex hull of P is $\mathcal{O}^2_\beta\mathcal{H}(P) = \mathbb{R}^2 - \bigcup_{w \in \mathcal{W}} w$.*

2 Computing and maintaining $\mathcal{O}^2_\beta\mathcal{H}(P)$

Based on Proposition 1, in order to compute $\mathcal{O}^2_\beta\mathcal{H}(P)$ we focus on the maximal stabbing $\mathcal{O}^2_\beta$-wedges containing no elements of P.

Moreover, for a point set P and a pair of lines $\mathcal{O}^2_\beta$ we define four staircase polygonal lines, as follows: The *top-right* β*-staircase* is the following sector of the boundary of the region obtained by removing from

*Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, alegria_c@uxmcc2.iimas.unam.mx.

†Departamento de Física y Matemáticas, Universidad de Alcalá, Spain, david.orden@uah.es. Partially supported by MICINN Project MTM2011-22792, ATLAS-Project VA172A12-2, and TIGRE5-CM Comunidad de Madrid Project S2013/ICE-2919.

‡Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain, carlos.seara@upc.edu. Partially supported by projects Gen. Cat. DGR 2014SGR46, MINECO MTM2012-30951/FEDER .

§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.

the plane all the top-right $\mathcal{O}^2_\beta$-wedges containing no element of P: It starts at the top element of P and ends at the element of P which is the rightmost one with respect to the non-horizontal line in $\mathcal{O}^2_\beta$. In a similar way we can define the top-left, bottom-left, and the bottom-right β-staircases of P.

In the sample $\mathcal{O}^2_\beta$-hull of Figure 1, the dotted lines are the directions of the oriented lines in $\mathcal{O}^2_\beta$ that are used as coordinate axes. Notice that the top-left β-staircase is just a point, and that $\mathcal{O}^2_\beta\mathcal{H}(P)$ is disconnected because of the intersections (the regions bounded by dashed lines) between the top-right and bottom-left β-staircases.

2.1 Maintaining the top-right staircase of P

We now show how to construct and maintain the top-right β-staircase of P as the angle β runs from 0 to π. Start by sorting, in $O(n \log n)$ time, the n points of P from bottom to top, and relabel and place them in this order in a list $\mathcal{L} = \{p_1, \ldots, p_n\}$.

For each p_i, $i = 2, \ldots, n-1$, compute the angles α^{a}_i, α^{b}_i, α^{c}_i, and α^{d}_i as shown in Figure 2. Note that for p_1 and p_n only two of these angles are defined. All these angles can be computed in $O(n)$ time. Notice that for an small enough initial value of β, all the elements of P belong to the top-right β-staircase of P and therefore, $\mathcal{O}^2_\beta\mathcal{H}(P) = P$.

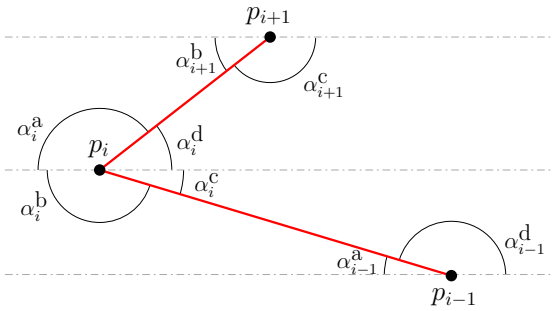


Figure 2: The four angles α^{a}_i, α^{b}_i, α^{c}_i, and α^{d}_i for the point p_i of P.

We observe next that, as the value of β increases, the first element of $\mathcal{L}$ to drop from the list is the p_i with the smallest angle α^d_i. Thus, when β reaches α^d_i, p_i leaves $\mathcal{L}$. Since p_i is no longer considered, we must update the angle of the predecessor p_{i-1} of p_i in $\mathcal{L}$ to be the angle between the horizontal line through p_{i-1} and the segment joining p_{i-1} to p_{i+1}. In a recursive way, if we have removed several elements of $\mathcal{L}$, the next element p_j to be eliminated is that with the smallest α^d_j. This can be obtained in logarithmic time using a priority queue. See Figure 3.

Hence, the total time complexity of calculating and maintaining the top-right β-staircase of P as β increases from 0 to π is $O(n \log n)$ and using linear space. At the end, when $\beta = \pi$, the only element remaining in $\mathcal{L}$ is the top point p_n.

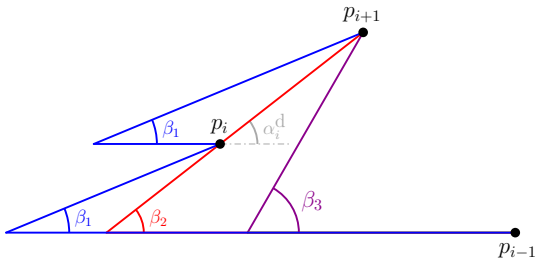


Figure 3: Portion of the top-right staircase for three values of β, before, at, and after the event $\beta = \alpha^{\mathrm{d}}_i$.

The top-left, bottom-left, and the bottom-right β-staircases of P can be computed and maintained in a similar way. The four β-staircases of P can be maintained simultaneously, as β goes from 0 to π, in $O(n \log n)$ time and $O(n)$ space.

Lemma 1 *$\mathcal{O}^2_\beta\mathcal{H}(P)$ for $\beta \in [0, \pi]$ can be maintained for $\beta \in [0, \pi]$ in $O(n \log n)$ time and $O(n)$ space.*

Proof. In order to maintain the boundary of $\mathcal{O}^2_\beta\mathcal{H}(P)$ for $\beta \in [0, \pi]$, apart from the four β-staircases of P we also need the sequence of the *overlap-events* which define when overlaps of $\mathcal{O}^2_\beta\mathcal{H}(P)$ finish. Initially, for β slightly greater than zero, consecutive points of P in the y-coordinate order define a very large overlap, whose area will decrease until reaching zero when the corresponding opposite wedges cease to intersect. In order to know when this happens, we need to maintain the current pairs of points which define the opposite wedges determining the overlap, updating them as in Figure 3, and focusing on the two points on rays with the direction of the non-horizontal line in $\mathcal{O}^2_\beta$. See Figure 5. The overlap finishes when β reaches the angle γ between the horizontal and the line through those two points.

The cost of this update is constant, once we know which point is to be changed. Nevertheless, we also need: (i) To maintain the list of the angles γ for all the current overlaps and (ii) To compute the minimum of this list, just to know which is the next overlap-event of ending-overlap. The cost of these updates is at most $O(\log n)$ time per insertion/deletion per point in P, each time such an overlap-event occurs. Since the number of these overlap-events is linear, the total cost is $O(n \log n)$ time. □

Standard techniques (refer to Chapter 4 in [6]) allow to obtain the boundary of $\mathcal{O}^2_\beta\mathcal{H}(P)$ in total $O(n \log n)$ time and $O(n)$ space. Furthermore, this time complexity of the algorithm is optimal, since given $\mathcal{O}^2_\beta\mathcal{H}(P)$ we can compute in linear time

$\mathcal{CH}(\mathcal{O}^2_\beta\mathcal{H}(P)) = \mathcal{CH}(P)$ and the computation of the usual convex hull $\mathcal{CH}(P)$ is in $\Omega(n \log n)$.

From the discussion above we get:

Theorem 2 *$\mathcal{O}^2_\beta\mathcal{H}(P)$ can be computed and maintained for $\beta \in [0, \pi]$ in $\Theta(n \log n)$ time and $O(n)$ space. The numbers of edges and connected components of $\mathcal{O}^2_\beta\mathcal{H}(P)$ for $\beta \in [0, \pi]$ can also be computed and maintained in the same running time and space.*

3 Optimizing the area and perimeter of $\mathcal{O}^2_\beta\mathcal{H}(P)$

Given an angle β, let the polygon $\mathcal{P}(\beta)$ be the one obtained joining counterclockwise consecutive vertices of the four staircases that define $\mathcal{O}^2_\beta\mathcal{H}(P)$.

Following the lines of Bae et al. [3], we express the area of $\mathcal{O}^2_\beta\mathcal{H}(P)$ in terms of the angle β, as

$$\text{area}(\mathcal{P}(\beta)) - \sum_i \text{area}(\triangle_i(\beta)) + \sum_j \text{area}(\parallelogram_j(\beta)),$$

where: (i) The triangles $\triangle_i$ are defined by a segment joining two consecutive vertices of a β-staircase S of P and the edges joining them along S. (ii) The parallelograms $\parallelogram_j$ are the overlaps between the boundaries of opposite staircases. See Figure 4.

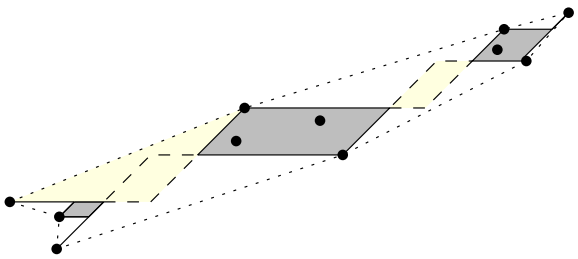


Figure 4: Dotted, the polygon $\mathcal{P}(\beta)$. In dark gray, the area of $\mathcal{O}^2\mathcal{H}_\beta(P)$. In yellow, a triangle and a parallelogram.

Next, we show how to compute each of the three terms in the formula. This allows us to get a general formula, which can be evaluated in each of the intervals $[\beta_i, \beta_{i+1}]$ between two consecutive events, obtaining the value of $\beta \in [\beta_i, \beta_{i+1}]$ which maximizes the area of $\mathcal{O}^2_\beta\mathcal{H}(P)$ in that interval. Note that there is a linear number of these intervals.

3.1 Polygon $\mathcal{P}(\beta)$

Observe that, as β increases from 0 to π, the set of vertices changes a linear number of times. This happens each time a point drops from one of the four staircases of P. Let $\mathcal{A} = \{\beta_1, \beta_2, \ldots, \beta_m\}$ be the set of angles at which the vertices of P drop out from the four staircases of P, $\beta_i < \beta_{i+1}$, $1 \le i \le m-1$.

Since the set of vertices of $\mathcal{P}(\beta)$ remains unchanged for any $\beta \in (\beta_i, \beta_{i+1})$, its area also remains unchanged. Thus, the area of $\mathcal{P}(\beta)$ has to be updated each time β reaches a value in $\mathcal{A}$. Since $\mathcal{A}$ has only a linear number of elements, the area of $\mathcal{P}(\beta)$ has to be updated a linear number of times. Each update can be done in constant time, as it involves the addition or subtraction of the areas of at most two triangles. See Figure 5. A flag will indicate when we have to add or to subtract.

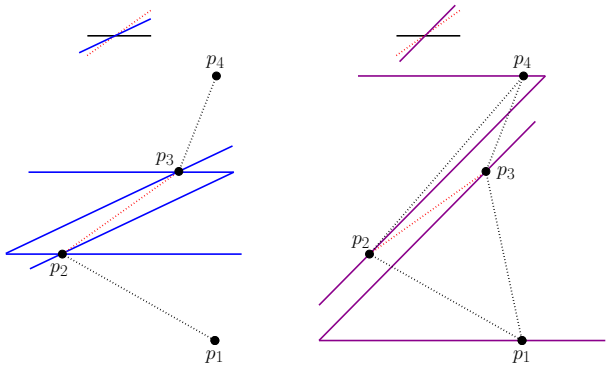


Figure 5: Left: Before the first event, the top-right and the bottom-left β-staircases of P are formed by all the points of P, the top-left β-staircase is the point p_4, and the bottom-right β-staircase is the point p_1. Right: After the first event, p_2 leaves the top-right β-staircase and p_3 leaves the bottom-left β-staircase.

3.2 Triangles $\triangle_i$

Since the number of vertices of $\mathcal{P}(\beta)$ changes only when β reaches a $\beta_i \in \mathcal{A}$, the number of triangles defined by $\mathcal{P}(\beta)$ also changes only when β equals some $\beta_i \in \mathcal{A}$.

Using elementary geometry, it can be checked that the sum of the areas of all the triangles of $\beta_i \in \mathcal{A}$ has the form $c + d\cot(\beta)$: It is sufficient to note that the area of each triangle $\triangle_i$ of $\mathcal{P}(\beta)$ has the form $|c_i \pm d_i \cot(\beta)|$. For example, if $p_i = (x_i, y_i)$ and $p_{i+1} = (x_{i+1}, y_{i+1})$ are consecutive vertices in the counterclockwise order of the top-right β-staircase of P, see Figure 6, then the area of the triangle $\triangle_i$ bounded by (i) the segment joining p_i to p_{i+1}, (ii) the horizontal line through p_i, and (iii) the line with angle β passing through p_{i+1} can be expressed as

$$\text{area}(\triangle_i) = |(x_i - x_{i+1})(y_{i+1} - y_i) + (y_{i+1} - y_i)^2 \cot(\beta)| =$$
$$= |c_i \pm d_i \cot(\beta)|.$$

3.3 Parallelograms $\parallelogram_j$

Parallelograms arise from overlaps between opposite β-staircases of the P. We need to compute the initial

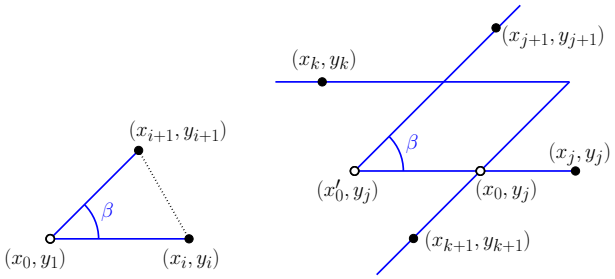


Figure 6: Left: Triangle corresponding to two consecutive points of the top-right β-staircase of P. Right: Parallelogram corresponding to two consecutive points of the top-right β-staircase and two consecutive points of the bottom-left β-staircase.

and final values of β for which each overlap is alive. These overlap events should be merged with the other events, in order to perform a discrete computation updating and computing the maximum values of the variables we want to optimize.

Overlaps can only arise between opposite staircases, that is between the top-right and the bottom-left β-staircases, or between the top-left and the bottom-right β-staircases.

Moreover, as β increases from 0 to π, all the overlaps between the top-left staircases and the bottom-right β-staircases arise after all the overlaps between the top-right and the bottom-left β-staircases. Thus we can process them independently, one after another.

The sum of the areas of these parallelograms can be expressed again as a function of the type $c' + d' \cot(\beta)$. For example, consider a parallelogram $\parallelogram_j$ determined by two consecutive points $p_j = (x_j, y_j)$ and $p_{j+1} = (x_{j+1}, y_{j+1})$ of the top-right β-staircase, together with two consecutive points $p_k = (x_k, y_k)$ and $p_{k+1} = (x_{k+1}, y_{k+1})$ of the bottom-left β-staircase. See Figure 6.

Note that the vertices of the parallelogram $\parallelogram_j$ are not p_j, p_{j+1}, p_k, and p_{k+1}. In fact, the parallelogram $\parallelogram_j$ is the intersection of two triangles of $\mathcal{P}(\beta)$, defined by p_j, p_{j+1}, and p_k, p_{k+1}.

Using elementary geometry, it can be checked that:

As β increases, the number of parallelograms generated by the top-right and the bottom-left β-staircases decreases. We need to compute in advance the events of the ends (and the beginnings) of overlaps, which are exactly the beginning (and the end) events of areas for the top-right and the bottom-left β-staircases the top-left and the bottom-right β-staircases.

Using again a priority queue, we can find the order in which the overlaps disappear in overall $O(n \log n)$ time. When a point defining an overlap changes, we have to update the corresponding area formula. This can be done in constant time.

The discussion above leads to the following result:

Theorem 3 *To compute the angle β such that $\mathcal{O}^2_\beta\mathcal{H}(P)$ has maximum area can be done in $O(n \log n)$ time and $O(n)$ space.*

As for maintaining the perimeter, it is enough to maintain the four staircases and the overlaps.

Thus, we get the following result:

Theorem 4 *To compute the angle β such that $\mathcal{O}^2_\beta\mathcal{H}(P)$ has minimum (non-zero) perimeter can be done in $O(n \log n)$ time and $O(n)$ space.*

Acknowledgments. The authors would like to thank the referees for their careful reading of this paper.

References

[1] C. Alegría-Galicia, T. Garduño, A. Rosas-Navarrete, C. Seara, and J. Urrutia. Rectilinear convex hull with minimum area. *XIV Spanish Meeting on Computational Geometry*, (2011). *LNCS*, Vol. 7579, (2012), 226–235.

[2] C. Alegría-Galicia, D. Orden, C. Seara, and J. Urrutia. On the $\mathcal{O}$-hull of planar point sets. *EuroCG 2014, Ein-Gedi, Israel, March 3–5, 2014.*

[3] S. W. Bae, C. Lee, H-K. Ahn, S. Choi, and K-Y. Chwa. Computing minimum-area rectilinear convex hull and L-shape. *Computational Geometry: Theory and Applications*, Vol. 42(3), (2009), 903–912.

[4] E. Fink and D. Wood. *Restricted-orientation convexity.* Springer, 2004.

[5] T. Ottman, E. Soisalon-Soisinen, and D. Wood. On the definition and computation of rectilinear convex hulls. *Information Sciences*, Vol. 33, (1984), 157–171.

[6] F. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, (1985).

[7] G. J. E. Rawlins. Explorations in Restricted-Orientation Geometry. PhD thesis, School of Computer Science, University of Waterloo, 1987.

A lower bound on opaque sets*

Akitoshi Kawamura† Sonoko Moriyama‡ Yota Otachi§ János Pach¶

Abstract

It is proved that the total length of any set of countably many rectifiable curves, whose union meets all straight lines that intersect the unit square U, is at least 2.00002. This is the first improvement on the lower bound of 2 established by Jones in 1964. A similar bound is proved for all convex sets U other than a triangle.

1 Introduction

A *barrier* or an *opaque set* for $U \subseteq \mathbb{R}^2$ is a set $B \subseteq \mathbb{R}^2$ that intersects every line that intersects U. For example, when U is a square, any of the four sets depicted in Figure 1 is a barrier. Note that some part of the barrier may lie outside U (Figure 2), and the barrier need not be connected. This notion dates back at least to Mazurkiewicz's work in 1916 [12].

We are interested in "short" barriers B for a given object U, and hence we restrict attention to *rectifiable* barriers B. By this we mean that B is a union of countably many curves b, pairwise disjoint except at the endpoints, that each have finite length $|b|$, and the sum of these lengths converges. We call this sum the *length* of B and denote it by $|B|$ (this does not depend on how we divide B into curves).

Finding the shortest barrier is hard, even for simple shapes U, such as the square, the equilateral triangle, and the disk [6, 10]. The shortest known barrier for the unit square is the last one in Figure 1, with length $2.638\ldots$. This problem and its relatives have an extensive literature. See [6, 11] and the introduction of [5] for more history, background, and related problems.

The best known lower bound for the unit square has been 2, established by Jones in 1964 [8]. In general, for convex U, a barrier needs to have length at least half the perimeter of U (we review a proof in Section 2):

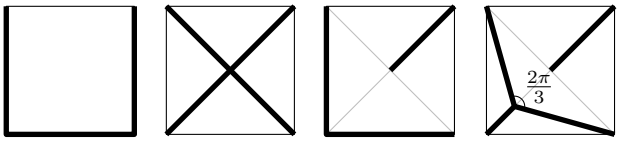


Figure 1: Barriers (in thick lines) for the unit square. The first one (three sides) and the second one (diagonals) have lengths 3 and $2\sqrt{2} = 2.828\ldots$, respectively. The third barrier consists of two sides and half of a diagonal, and has length $2 + 1/\sqrt{2} = 2.707\ldots$. The last one is the shortest known barrier, with length $\sqrt{2} + \sqrt{6}/2 = 2.638\ldots$, consisting of half a diagonal and the Steiner tree of the lower left triangle.

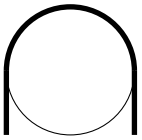


Figure 2: A barrier (in thick lines) for a disk that is shorter than the perimeter. This is not the shortest one; see [6].

Lemma 1 *$|B| \geq p$ for any rectifiable barrier B of a convex set $U \subseteq \mathbb{R}^2$ with perimeter $2p$.*

Thus, from the point of view of finding short barriers, the trivial strategy of enclosing the entire perimeter (or the perimeter of the convex hull if U is a non-convex connected set) gives a 2-approximation. See [4] and references therein for algorithms that find shorter barriers. The current best approximation ratio is $1.58\ldots$ [5].

Proving a better lower bound has been elusive (again, even for specific shapes U). There has been some partial progress under additional assumptions about the shape (single arc, connected, etc.) and location (inside U, near U, etc.) of the barrier [1, 3, 7, 11, 14], but establishing an unconditional lower bound strictly greater than 2 for the unit square has been open (see [4, Open Problem 5] or [3, Footnote 1]). We obtained such a lower bound:

Theorem 2 *$|B| \geq 2.00002$ for any rectifiable barrier B of the unit square $\square$.*

Dumitrescu and Jiang [3] recently obtained a lower

*A full version is available [9]. The work presented here was supported in part by JSPS KAKENHI, by the ELC project (Grant-in-Aid for Scientific Research on Innovative Areas, MEXT, Japan), by OTKA under EUROGIGA projects GraDR and ComPoSe 10-EuroGIGA-OP-003, and by Swiss National Science Foundation Grants 200020-144531 and 200021-137574.

†University of Tokyo
‡Nihon University
§Japan Advanced Institute of Science and Technology
¶EPFL, Lausanne and Rényi Institute, Budapest

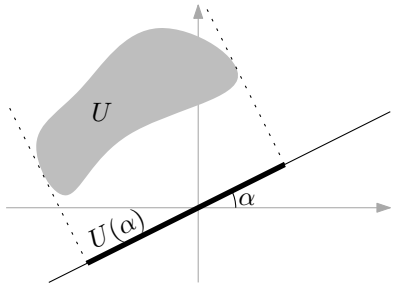


Figure 3: The image $U(\alpha) \subseteq \mathbb{R}$ of U.

bound of $2 + 10^{-12}$ under the assumption that the barrier lies in the square obtained by magnifying $\square$ by 2 about its centre. Their proof, conceived independently of ours and at about the same time, is based on different ideas, most notably the line-sweeping technique. It will be worth exploring whether their techniques can be combined with ours.

Our proof can be generalized:

Theorem 3 *For any closed convex set U with perimeter $2p$ that is not a triangle, there is $\varepsilon > 0$ such that any barrier B for U has length at least $p + \varepsilon$.*

See the full version [9] for a proof. Thus, the only convex objects for which we fail to establish a lower bound better than Lemma 1 are triangles.

Due to space constraint, we will only sketch our proof of Theorem 2 in Section 3. In the final section, we discuss a closely related question.

2 Preliminaries: A general lower bound

For a set U and an angle $\alpha \in [0, 2\pi)$ (all angle calculation will be performed modulo 2π), we write $U(\alpha) \subseteq \mathbb{R}$ for the image of U projected onto the line passing through the origin and enclosing angle $+\alpha$ with the positive x-axis, i.e.,

$$U(\alpha) = \left\{ x \cos\alpha + y \sin\alpha : (x, y) \in U \right\} \qquad (1)$$

(Figure 3). To say that B is a barrier of U means that $B(\alpha) \supseteq U(\alpha)$ for all α.

For the discussion of upper and lower bounds on the length of a barrier, the following lemma says that it suffices to consider barriers that are a countable union of line segments. We call such a barrier *straight*.

Lemma 4 ([5, Lemma 1]) *Let B be a rectifiable barrier for $U \subseteq \mathbb{R}^2$. Then, for any $\varepsilon > 0$, there exists a straight barrier B_ε for U such that $|B_\varepsilon| \leq (1+\varepsilon)|B|$.*

Since the proof in [5] has a gap, we include another proof in our full version [9].

As mentioned in the introduction (Lemma 1), it has been known that any barrier of a convex set must be at least half the perimeter. We include a short proof of this bound here, for completeness and further reference. See [2] for another elegant proof.

Proof. [Proof of Lemma 1] By Lemma 4, we may assume that B consists of line segments. We have

$$\begin{aligned} |U(\alpha)| &\leq |B(\alpha)| \\ &\leq \sum_b |b(\alpha)| = \sum_b |b| \cdot |\cos(\alpha - \theta_b)| \end{aligned} \qquad (2)$$

for each $\alpha \in [0, 2\pi)$, where the sum is taken over all line segments b that comprise B without overlaps, and θ_b is the angle of b. Integrating over $[0, 2\pi)$, we obtain

$$\begin{aligned} \int_{\alpha=0}^{2\pi} |U(\alpha)| \, \mathrm{d}\alpha &\leq \sum_b \left(|b| \cdot \int_{\alpha=0}^{2\pi} |\cos(\alpha - \theta_b)| \, \mathrm{d}\alpha \right) \\ &= 4 \sum_b |b| = 4|B|. \end{aligned} \qquad (3)$$

When U is a convex set, the left-hand side equals twice the perimeter (cf. the Cauchy–Crofton formula [13, Theorem 16.15]). $\square$

3 Proof ideas for Theorem 2

Note that Theorems 2 and 3 do not merely state the non-existence of a straight barrier B of length exactly half the perimeter of U. Such a claim can be proved easily as follows: If B is such a barrier, the inequality (3) must hold with equality, and so must (2) for each α. Thus, the second inequality in (2) must hold with equality, which means that B never overlaps with itself when projected onto the line with angle α. Since this must be the case for all α, the entire B must lie on a line, which is clearly impossible.

The theorems claim more strongly that a barrier must be longer by an absolute constant. The following lemma says that in order to obtain such a bound, we should find a part $B' \subseteq B$ of the barrier whose contribution to covering U is less than the optimal by at least a fixed positive constant.

Lemma 5 *Let B be a barrier of a convex polygon U of perimeter $2p$. Then $|B| \geq p + \delta$ if there is a subset $B' \subseteq B$ with*

$$\int_{\alpha=0}^{2\pi} |B'(\alpha) \cap U(\alpha)| \, \mathrm{d}\alpha \leq 4|B'| - 4\delta. \qquad (4)$$

Proof. For each $\alpha \in [0, 2\pi)$, we have $U(\alpha) \subseteq B(\alpha)$, and thus

$$\begin{aligned} |U(\alpha)| &= |B(\alpha) \cap U(\alpha)| \\ &\leq |(B \setminus B')(\alpha) \cap U(\alpha)| + |B'(\alpha) \cap U(\alpha)| \\ &\leq |(B \setminus B')(\alpha)| + |B'(\alpha) \cap U(\alpha)|. \end{aligned} \qquad (5)$$

Integrating over $\alpha \in [0, 2\pi)$ and using the assumption (4), we get $4p \leq 4|B \setminus B'| + (4|B'| - 4\delta) = 4|B| - 4\delta$. $\square$

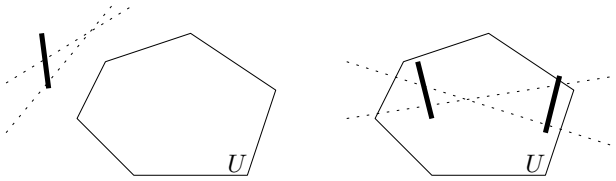


Figure 4: Two wasteful situations. Left: a barrier segment (thick) lies far outside the object U, which causes significant waste because this segment covers in vain some lines (dotted) that do not meet U; this is discussed in Lemma 6. Right: two parts of the barrier (thick) are facing each other, which is also wasteful because they cover some lines (dotted) doubly; this is roughly the situation discussed in Lemma 7.

There are several ways in which such a "waste" can occur, and we make use of two of them (Figure 4). The first one is when there is a significant part of the barrier that lies far outside U, as described in the following lemma (see the full version [9] for a proof, which is relatively straightforward):

Lemma 6 *Let b be a line segment that lies outside a convex region U. Suppose that the set $A := \{\,\alpha \in [0, 2\pi) : U(\alpha) \cap b(\alpha) \neq \emptyset\,\}$ (of angles of all lines through U and b) has measure $\leq 2\pi - 4\varepsilon$. Then*

$$\int_{\alpha=0}^{2\pi} |b(\alpha) \cap U(\alpha)|\,\mathrm{d}\alpha \leq 4|b|\cos\varepsilon. \tag{6}$$

The second situation where we have a significant waste required in Lemma 5 is when there are two sets of barrier segments that roughly face each other:

Lemma 7 *Let $\lambda \in (0, \frac{\pi}{2})$, $\kappa \in (0, \lambda)$ and l, $D > 0$. Let B^- and B^+ be unions of n line segments of length l (Figure 5) such that*

1. *every segment of $B^- \cup B^+$ makes angle $> \lambda$ with the horizontal axis;*
2. *$B^- \cup B^+$ lies entirely in the disk of diameter D centred at the origin;*
3. *B^- and B^+ are separated by bands of angle κ and width $W := nl\sin(\lambda - \kappa)$ centred at the origin, as depicted in Figure 5—that is, each point $(x, y) \in B^\pm$ satisfies $\pm(x\sin\kappa + y\cos\kappa) \geq W/2$ and $\pm(x\sin\kappa - y\cos\kappa) \geq W/2$ (where $\pm$ should be read consistently as $+$ and $-$).*

Then

$$\int_{\alpha=0}^{2\pi} |(B^- \cup B^+)(\alpha)|\,\mathrm{d}\alpha \leq 8nl - \frac{2W^2}{D}. \tag{7}$$

Note that $8nl = 4|B^- \cup B^+|$, so (7) is of the form (4) in Lemma 5. Using Lemmas 6 and 7, our proof of Theorem 2 roughly goes as follows. Consider a barrier whose length is very close to 2.

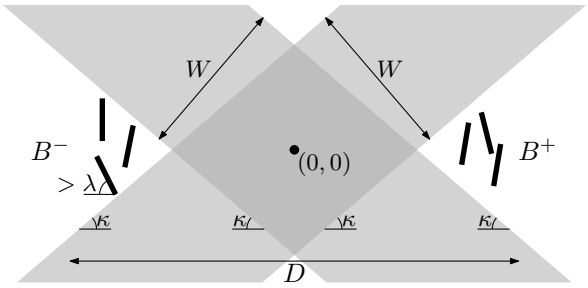


Figure 5: Sets B^- and B^+ (Lemma 7).

1. There cannot be too much of the barrier far outside $\square$, because that would be too wasteful by Lemma 6.
2. This implies that there must be a significant part of the barrier near each vertex of $\square$, because this is the only place to put barrier segments that block lines that intersect $\square$ only near this vertex.
3. Among the parts of the barrier that lie near the four vertices, there are parts that face each other and thus lead to waste by Lemma 7. (It is this last step that requires *four* corners and thus prevents us from proving the generalized Theorem 3 for triangles: when we have only three corners, it can happen that most barrier segments near the first, second and third corners, respectively, point towards the second, third and the first corners, and thus no two of them face each other.)

See the full version [9] for a complete proof.

We remark that Lemma 7 requires a much more involved argument than Lemma 6. Here we only comment on what Lemma 7 claims intuitively and what makes it nontrivial to prove. By symmetry, we can halve the interval $[0, 2\pi]$ and see that (7) is equivalent to

$$4nl - \int_{\alpha=0}^{\pi} |(B^- \cup B^+)(\alpha)|\,\mathrm{d}\alpha \geq \frac{W^2}{D}. \tag{8}$$

Let $\mathcal{B}^-$ and $\mathcal{B}^+$ be the sets of line segments of length l comprising B^- and B^+, respectively. For each $b \in \mathcal{B}^- \cup \mathcal{B}^+$, consider the region

$$R_b := \{\,(\alpha, v) \in [0, \pi] \times \mathbb{R} : v \in b(\alpha)\,\}, \tag{9}$$

whose area is $2l$. Note that the first term $4nl$ of (8) is the sum of this area for all $b \in \mathcal{B}^- \cup \mathcal{B}^+$, whereas the second term is the area of the union. Thus, (8) says that the area of the overlap (considering multiplicity) is at least W^2/D. Since this term W^2/D is proportional to n^2, which is the number of pairs $(b, b') \in \mathcal{B}^- \times \mathcal{B}^+$, we should lower-bound (by a constant determined by λ, κ, D) the area of the overlap $R_b \cap R_{b'}$ per such pair (b, b'). This is relatively easy if the overlaps $R_b \cap R_{b'}$ are all disjoint, but it can get tricky otherwise. See the full version [9] for details.

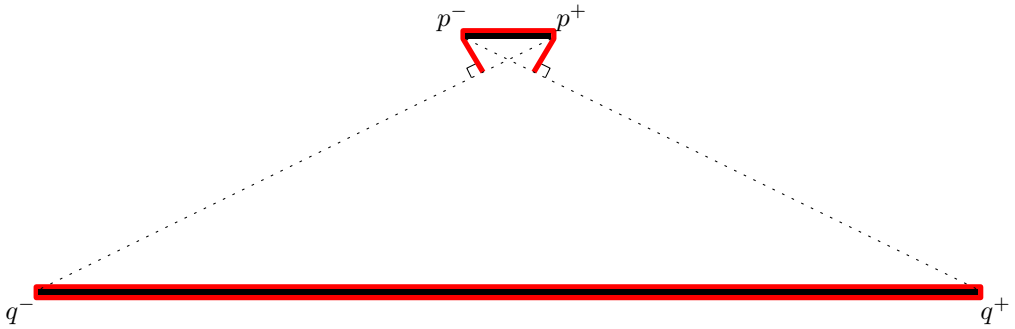


Figure 6: Consider the line segments p^-p^+ and q^-q^+, where $p^\pm = (\pm 1, 8)$ and $q^\pm = (\pm 15, 0)$, and let U be the union of these segments with small "thickness": U consists of a rectangle with vertices $(\pm 1, 8 \pm \varepsilon)$ and another with vertices $(\pm 15, \pm\varepsilon)$, for a small $\varepsilon > 0$. The boundaries of these thick line segments have total length 64 (plus a small amount due to the thickness). The boundary of the convex hull of all of U has length $2 + 30 + 2\sqrt{260} > 64.24$ (plus thickness). But we have another half-line barrier depicted above, whose total length is $2 + 60 + 2/\sqrt{5} + 2/\sqrt{5} < 63.79$ (plus thickness, which can be made arbitrarily small).

4 Half-line barriers

We propose an analogous question, obtained by replacing lines by half-lines in the definition of barriers: a set $B \subseteq \mathbb{R}^2$ is a *half-line barrier* of $U \subseteq \mathbb{R}^2$ if all half-lines intersecting U intersect B. This intuitively means "hiding the object U from outside," which we find perhaps as natural, if not more, than the notion of opaque sets. Similarly to Lemma 1, we have

Lemma 8 *$|B| \geq p$ for any rectifiable half-line barrier B of a convex set $U \subseteq \mathbb{R}^2$ that is not a line segment and has perimeter p.*

Thus, unlike for line barriers, the question is completely answered when U is connected: the shortest half-line barrier is the boundary of the convex hull.

If U is disconnected, there can be shorter half-line barriers. For example, if U consists of two connected components that are far apart from each other, it is more efficient to cover them separately than together. One might hope that an optimal half-line barrier is always obtained by grouping the connected components of U in some way and taking convex hulls of each. This is not true, as Figure 6 shows. We have not been able to find an algorithm that achieves a nontrivial approximation ratio for this problem.

Acknowledgments We are grateful to Gábor Tardos for many interesting discussions on the subject.

References

[1] H. T. Croft. Curves intersecting certain sets of great-circles on the sphere. *Journal of the London Mathematical Society (2)*, 1, 461–469, 1969.

[2] E. Demaine and J. O'Rourke. Open problems from CCCG 2007. In *Proc. 20th Canadian Conference on Computational Geometry* (CCCG 2008), 183–186, 2008.

[3] A. Dumitrescu and M. Jiang. The opaque square. In *Proc. 30th Annual Symposium on Computational Geometry* (SoCG 2014), 529–538, 2014.

[4] A. Dumitrescu and M. Jiang. Computational Geometry Column 58. *SIGACT News*, 44(4), 73–78, 2013.

[5] A. Dumitrescu, M. Jiang, and J. Pach. Opaque sets. *Algorithmica*, in press. Preprint: arXiv:1005.2218v5.

[6] V. Faber and J. Mycielski. The shortest curve that meets all the lines that meet a convex body. *American Mathematical Monthly*, 93, 796–801, 1986.

[7] V. Faber, J. Mycielski, and P. Pedersen. On the shortest curve which meets all the lines which meet a circle. *Annales Polonici Mathematici*, 44, 249–266, 1984.

[8] R. E. D. Jones. Opaque sets of degree α. *American Mathematical Monthly*, 71, 535–537, 1964.

[9] A. Kawamura, S. Moriyama, Y. Otachi, and J. Pach. A lower bound on opaque sets. Preprint: arXiv:1403.3894.

[10] B. Kawohl. The opaque square and the opaque circle. *General Inequalities 7*, ISNM International Series of Numerical Mathematics Volume 123, 339–346, 1997.

[11] B. Kawohl. Some nonconvex shape optimization problems. In *Optimal Shape Design*, 7–46, Springer, 2000.

[12] S. Mazurkiewicz. Przykład zbioru domkniętego, punktokształtnego, mającego punkty wspólne z każdą prostą, przecinającą pewien obszar domknięty. (Sur un ensemble fermé, punctiforme, qui rencontre toute droite passant par un certain domaine.) *Prace Matematyczno-Fizyczne*, 27, 11–16, 1916. In Polish (French summary).

[13] J. Pach and P. K. Agarwal. *Combinatorial Geometry*. Wiley, New York, 1995.

[14] J. S. Provan, M. Brazil, D. Thomas, J. F. Weng. Minimum opaque covers for polygonal regions. Preprint: arXiv:1210.8139v1.

Approximating the Colorful Carathéodory Theorem

Wolfgang Mulzer* Yannik Stein*

Abstract

Let $P_1, \ldots, P_{d+1} \subset \mathbb{R}^d$ be d-dimensional point sets such that the convex hull of each P_i contains the origin. We call the sets P_i *color classes*, and we think of the points in P_i as having color i. A *colorful choice* is a set with at most one point from each color class. The *colorful Carathéodory theorem* guarantees the existence of a colorful choice whose convex hull contains the origin. So far, the computational complexity of finding such a colorful choice is unknown.

An *m-colorful choice* is a set that contains at most m points from each color class. We present an approximation algorithm that computes for any constant $\varepsilon > 0$, an $\lceil \varepsilon(d+1) \rceil$-colorful choice containing the origin in its convex hull in polynomial time. This notion of approximation has not been studied before, and it is motivated through the applications of the colorful Carathéodory theorem in the literature. Second, we show that the exact problem can be solved in $d^{O(\log d)}$ time if $\Theta(d^2 \log d)$ color classes are available, improving over the trivial $d^{O(d)}$ time algorithm.

1 Introduction

Let $P \subset \mathbb{R}^d$ be a point set. Carathéodory's theorem [4, Theorem 1.2.3] states that if $\vec{0} \in \operatorname{conv}(P)$, there is a subset $P' \subseteq P$ of size $d+1$ with $\vec{0} \in \operatorname{conv}(P')$. Bárány [1] gives a *colorful* generalization.

Theorem 1 (Colorful Carathéodory Theorem) *Let $P_1, \ldots, P_{d+1} \subset \mathbb{R}^d$ be point sets (the color classes) with $\vec{0} \in \operatorname{conv}(P_i)$, for $i = 1, \ldots, d+1$. There is a* colorful choice *C with $\vec{0} \in \operatorname{conv}(C)$. Here, a colorful choice is a set with at most one point from each color class.*

Theorem 1 yields Carathéodory's theorem by setting $P_1 = \cdots = P_{d+1}$. Moreover, there are many variants with weaker assumptions [5]. While Carathéodory's theorem has a proof that gives a polynomial-time algorithm, very little is known about the algorithmic complexity of the colorful Carathéodory theorem [2]. This question is particularly interesting since Sarkaria's proof [10] of Tverberg's theorem [11] can be interpreted as a polynomial-time reduction from computing Tverberg partitions to computing a colorful choice with the origin in its convex hull. Both problems lie in *Total Function NP* (TFNP), the complexity class of total search problems that are solvable in non-deterministic polynomial time. It is well known that no problem in TFNP is NP-hard unless NP = coNP [3].

Related problems have been shown to be complete for subclasses of TFNP. Recently, Meunier and Sarrabezolles [6] proved that given $d+1$ pairs of points $P_1, \ldots, P_{d+1} \in \mathbb{Q}^d$ and a colorful choice that contains the origin in its convex hull, it is PPAD-complete [9] to find another colorful choice with the origin in its convex hull. The authors [8] showed the following generalization of the colorful Carathéodory problem to be PLS-complete [3]: given sets $P_1, \ldots, P_n \subset \mathbb{R}^d$, find a colorful choice s.t. the distance of its convex hull to the origin cannot be decreased by swapping a *single* point with another point of the same color.

Since we have no polynomial-time algorithms for the colorful Carathéodory theorem, approximation algorithms are of interest. This was first studied by Bárány and Onn [2] who described how to find a colorful choice whose convex hull is "close" to the origin. Let $\varepsilon, \rho > 0$ be parameters. Given sets $P_1, \ldots, P_{d+1} \in \mathbb{Q}^d$ encoded in L bits s.t. (i) each P_i contains a ball of radius ρ centered at the origin in its convex hull; and (ii) all points $p \in P_i$ fulfill $1 \leq \|p\| \leq 2$, one can find a colorful choice C s.t. $d(\vec{0}, \operatorname{conv}(C)) \leq \varepsilon$ in time $\operatorname{poly}(L, \log(\varepsilon^{-1}), \rho^{-1})$ on the Word-Ram with logarithmic costs. If $\rho^{-1} = L^{O(1)}$, the algorithm actually guarantees $\vec{0} \in \operatorname{conv}(C)$.

However, when using the colorful Carathéodory theorem in a proof, it is often crucial that the colorful choice contains the origin in its convex hull. Being "close" is not enough. On the other hand, allowing multiple points from each color class may have a natural interpretation in the reduction. This is the case in Sarkaria's proof [10] of Tverberg's theorem and in the proof of the First Selection Lemma [4, Theorem 9.1.1]. This motivates a different notion of approximation. Given a parameter m and sets $P_1, \ldots, P_{d+1} \in \mathbb{Q}^d$, find a set C s.t. $\vec{0} \in \operatorname{conv}(C)$ and s.t. $|C \cap P_i| \leq m$ for all P_i. In contrast to Bárány and Onn's setting, we have no general position assumption. Surprisingly, this notion does not seem to have been studied before.

*Institut für Informatik, Freie Universität Berlin, Germany. {mulzer,yannikstein}@inf.fu-berlin.de. W. Mulzer is partially supported by DFG Grants MU 3501/1 and MU 3501/2. Y. Stein is supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

Our Results. Given sets $P_1, \ldots, P_n \subset \mathbb{R}^d$, we call a set C containing at most m points from each set P_i an *m-colorful choice*. A 1-colorful choice is also called *perfect colorful choice*. All presented algorithms are analyzed on the Real-Ram model with unit costs. We begin with an algorithm based on a dimension reduction argument that repeatedly combines approximations for lower dimensional linear subspaces. This leads to the following result:

Theorem 2 *Let $P_1, \ldots, P_{d+1} \subset \mathbb{R}^d$ be sets of size at most $d+1$ s.t. $\vec{0} \in \operatorname{conv}(P_i)$ for $i = 1, \ldots, d+1$. Then, for any $\varepsilon = \Omega(d^{-1/3})$, an $\lceil \varepsilon(d+1) \rceil$-colorful choice containing the origin in its convex hull can be computed in $d^{O((1/\varepsilon)\log(1/\varepsilon))}$ time.*

In particular, for any constant ε the algorithm from Theorem 2 runs in polynomial-time. Given $\Theta(d^2 \log d)$ color classes, we can also improve the naive $d^{O(d)}$ algorithm for finding a perfect colorful choice.

Theorem 3 *Let $P_1, \ldots, P_n \subset \mathbb{R}^d$ be $n = \Theta(d^2 \log d)$ sets of size at most $d+1$ s.t. $\vec{0} \in \operatorname{conv}(P_i)$, for $i = 1, \ldots, n$. Then, a perfect colorful choice can be computed in $d^{O(\log d)}$ time.*

2 Fundamentals

Throughout the paper, we denote for a given point set $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ by $\operatorname{span}(P) = \{\sum_{i=1}^n \alpha_i p_i \mid \alpha_i \in \mathbb{R}\}$ its linear span and by $\operatorname{span}(P)^\perp = \{v \in \mathbb{R}^d \mid \forall p \in \operatorname{span}(P) : \langle v, p \rangle = 0\}$ the subspace orthogonal to $\operatorname{span}(P)$; by $\operatorname{aff}(P) = \{\sum_{i=1}^n \alpha_i p_i \mid \alpha_i \in \mathbb{R}, \sum_{i=1}^n \alpha_i = 1\}$ its affine hull; by $\operatorname{pos}(P) = \{\sum_{i=1}^n \mu_i p_i \mid \mu_i \geq 0\}$ all linear combinations with nonnegative coefficients; by $\operatorname{conv}(P) = \{\sum_{i=1}^n \lambda_i p_i \mid \lambda_i \geq 0, \sum_{i=1}^n \lambda_i = 1\}$ its convex hull; and by $\dim(P)$ the dimension of $\operatorname{span}(P)$.

Furthermore, we say that a set $P \subset \mathbb{R}^d$ is in *general position* if for every $k \leq d$, no $k+2$ points lie in a k-flat *and* if no proper subset of P contains the origin in its convex hull. We also use the following constructive version of Carathéodory's theorem:

Lemma 4 *Let $P \subset \mathbb{R}^d$ be a set of $O(d)$ points s.t. $\vec{0} \in \operatorname{conv}(P)$. In $O(d^4)$ time, we can find a subset $P' \subseteq P$ of at most $d+1$ points in general position such that P' contains the origin in its convex hull.* □

3 Approximation by Rebalancing

Let $P_1, \ldots, P_{d+1} \subset \mathbb{R}^d$ be the color classes and $\lceil \varepsilon(d+1) \rceil$ be the desired approximation guarantee. Throughout the algorithm, we maintain a temporary approximation $C \subset P_1 \cup \cdots \cup P_{d+1}$ that contains the origin in its convex hull, but may have more than $\lceil \varepsilon(d+1) \rceil$ points of the same color. The algorithm then repeatedly replaces at least one point from each color class that appears more than $\lceil \varepsilon(d+1) \rceil$ times in C by colors that appear only "few" times using a dimension reduction argument.

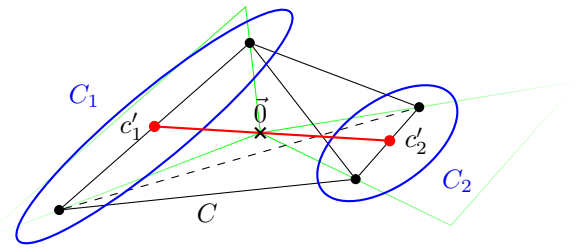


Figure 1: Example of Lemma 6 in three dimensions.

The following lemma enables us to replace a single point in C by an approximate colorful choice for the orthogonal space $\operatorname{span}(C)^\perp$:

Lemma 5 *Let $C \subset \mathbb{R}^d$, $|C| = k \leq d+1$, be a set in general position that contains the origin in its convex hull. Furthermore, let $Q \subset \mathbb{R}^d$ be a set of size $O(d)$ whose orthogonal projection onto $\operatorname{span}(C)^\perp$ contains the origin in its convex hull. Then, there is a point $c \in C$ computable in $O(d^4)$ time s.t. $\vec{0} \in \operatorname{conv}(Q \cup C \setminus \{c\})$.*

Proof. Write $Q = \{q_1, \ldots, q_l\}$. Each q_i can be expressed as $\widetilde{q}_i + \hat{c}_i$, where $\widetilde{q}_i$ denotes the orthogonal projection of q_i onto $\operatorname{span}(C)^\perp$ and $\hat{c}_i \in \operatorname{span}(C)$. By our assumption, the origin is a convex combination of $\widetilde{q}_1, \ldots, \widetilde{q}_l$: $\vec{0} = \sum_{i=1}^l \lambda_i \widetilde{q}_i$, where $\lambda_i \geq 0$ and $\sum_{i=1}^l \lambda_i = 1$. Consider the convex combination $q = \sum_{i=1}^l \lambda_i q_i$ of points in Q with the same coefficients. Since $q = \sum_{i=1}^l \lambda_i q_i = \sum_{i=1}^l \lambda_i (\widetilde{q}_i + \hat{c}_i) = \sum_{i=1}^l \lambda_i \hat{c}_i$, q is contained in $\operatorname{span}(C)$. By our assumption, we have $\vec{0} \in \operatorname{conv}(C)$ and C is in general position. It can be easily verified that this implies $\operatorname{pos}(P) = \operatorname{span}(C)$. Thus, there are $k-1$ points $c_{j_1}, \ldots, c_{j_{k-1}}$ in C s.t. $-q \in \operatorname{pos}(c_{j_1}, \ldots, c_{j_{k-1}})$. We take $c \in C$ as the single point that does not appear in $c_{j_1}, \ldots, c_{j_{k-1}}$. It can be found in $O(d^4)$ time by solving $k \leq d+1$ linear systems of equations $L_1, \ldots, L_k$, where L_j is defined as $\sum_{c_i \in C, i \neq j} \alpha_i c_i = -q$. Since C is in general position, all $(k-1)$-subsets of C are a basis for $\operatorname{span}(C)$. Thus, the linear systems have unique solutions. Since $\vec{0} \in \operatorname{conv}(C)$, one of the linear systems has a solution with no negative coefficients. □

Unfortunately, we do not know how to influence the color of c in Lemma 5. We would like to replace a point whose color contributes more than $\lceil \varepsilon(d+1) \rceil$ points to C. The next lemma gives us more control.

Lemma 6 *Let $C \subset \mathbb{R}^d$, $|C| \leq d+1$, be a set in general position s.t. $\vec{0} \in \operatorname{conv}(C)$ and let $C_1, \ldots, C_m$ be a partition of C. Then, we can find in $O(d^3)$ time a set $C' = \{c_1', \ldots, c_m'\} \subset \mathbb{R}^d$ with the following properties:*

(1) $\forall i = 1, \ldots, m$: $c'_i \in \text{pos}(C_i) \setminus \{\vec{0}\}$; *(2)* $\vec{0} \in \text{conv}(C')$; *and (3)* $\dim(C') = m - 1$. *We call the points in* C' representatives *for* C *w.r.t. the partition* $C_1, \ldots, C_m$.

Proof. Since C contains the origin in its convex hull, we can write $\vec{0}$ as $\vec{0} = \sum_{c \in C} \lambda_c c$, where all $\lambda_c > 0$, since C is in general position. Define c'_j as $c'_j = \sum_{c \in C_j} \lambda_c c$ for $i = 1, \ldots, m$. Properties 1. and 2. can be easily verified for the set $C' = \{c'_1, \ldots, c'_m\}$. Furthermore, c'_1 can be expressed as a linear combination of the other points in C': $c'_1 = -(c'_2 + \cdots + c'_m)$. Thus, $\dim(C') < m$. On the other hand, we have $\dim(C') \geq m - 1$ due to general position. This proves Property 3. See Figure 1 for an example. □

Instead of applying Lemma 5 to C directly, we use Lemma 6 to obtain a carefully chosen set of representative points and apply Lemma 5 to replace a representative. By choosing the partition for the representatives appropriately, we can influence the color of the removed points.

Now, we are ready to put everything together. The algorithm repeatedly replaces points in C by a recursively computed approximate colorful choice for a linear subspace. We are given as input the color classes $P_1, \ldots, P_{d+1} \subset \mathbb{R}^d$, each containing the origin in its convex hull, and the current recursion depth j. Define $\mathcal{M}(j)$ as $\mathcal{M}(j) = \lceil (1-\varepsilon)^{-j/2} \varepsilon (d+1) \rceil$ and $\mathcal{D}(j)$ as $\mathcal{D}(j) = \lceil (1-\varepsilon)^{j} \varepsilon (d+1) \rceil$. In recursion level j, the input is $\mathcal{D}(j)$-dimensional and the algorithm computes an $\mathcal{M}(j)$-colorful choice. Hence, in the topmost recursion level (i.e., $j = 0$), a $\lceil \varepsilon(d+1) \rceil$-colorful choice is computed. If $d = O(1)$, we compute an approximation by brute force. Otherwise, we initialize the temporary approximation C with a complete color class and prune it with Lemma 4. If the pruned set C is an $\mathcal{M}(j)$-colorful choice, we return it. Otherwise, we repeat the following balancing-steps: we partition C into $k = \mathcal{D}(j) - \mathcal{D}(j+1) + 1$ sets $C_1, \ldots, C_k$, where the points from each color in C are distributed evenly among the k sets. Let $n_i = |P_i \cap C|$ denote the number of points from P_i in C. Since $k \leq \mathcal{M}(j) + 1$, each set in the partition contains at least one point from each color class P_i for which $n_i \geq \mathcal{M}(j) + 1$. Applying Lemma 6, we compute representatives $C' = \{c'_1, \ldots, c'_k\}$ for this partition. Note that $\dim(C') = k - 1$ and that $\dim(\text{span}(C')^{\perp}) = \mathcal{D}(j) - k + 1 = \mathcal{D}(j+1)$. We call a color class P_i *light* if $n_i \leq \mathcal{M}(j) - \mathcal{M}(j+1)$, and *heavy*, otherwise. We find $d - k + 2$ light color classes and project them orthogonally onto $\text{span}(C')^{\perp}$. Let $\widetilde{P}_{j_1}, \ldots, \widetilde{P}_{j_{d-k+2}}$ denote the projections. Next, we recursively compute an $\mathcal{M}(j+1)$-colorful choice $\widetilde{Q}$ for the space orthogonal to $\text{span}(C')$ with $(\widetilde{P}_{j_1}, \ldots, \widetilde{P}_{j_{\mathcal{D}(j+1)+1}}, j+1)$ as input. Let Q be the point set whose projection gives $\widetilde{Q}$. Using Lemma 5, we compute a point $c'_j \in C'$ s.t. $\text{conv}(Q \cup C' \setminus c'_j)$ contains the origin. We replace the subset C_j of C by Q and prune C again with Lemma 4. Since each representative c'_i is contained in the cone $\text{pos}(C_i)$, $Q \cup C \setminus C_j$ still contains the origin in its convex hull and the invariant is maintained. Thus, in each iteration, at least one point from each color class P_i for which $n_i > \mathcal{M}(j)$ is replaced by points from light color classes. This is repeated until no color class appears more than $\mathcal{M}(j)$ times in C.

Proof of Theorem 2. We prove correctness by induction on the recursion depth. In particular, we show that the input in the jth recursion is $\mathcal{D}(j)$-dimensional and that an $\mathcal{M}(j)$-colorful choice is returned. There are two base cases: if $d = O(1)$ we compute a perfect colorful choice by brute force in $O(1)$ time. This is always a valid approximation regardless of $\mathcal{M}$. If $\mathcal{D}(j) + 1 \leq \mathcal{M}(j)$, we obtain a valid approximation by pruning C with Lemma 4. Hence, the claim holds in both base cases. In each level of the recursion, the dimension is reduced by $k - 1$. The dimension of the input in the recursion is thus $\mathcal{D}(j) - k - 1 = \mathcal{D}(j+1)$ as claimed. Since $\mathcal{D}$ is decreasing, some base case is reached eventually. Assume now that the current recursion depth is j and that the claim holds for all $j' > j$. Let $C^{(t)}$ denote the set C after t iterations of the balancing-steps in the jth recursion and $n_i^{(t)}$ the number of points from P_i in C. Define the *excess of a color* P_i as $e_i^{(t)} = \max\{0, n_i^{(t)} - \mathcal{M}(j)\}$ and the *excess of* $C^{(t)}$ as $\mathcal{E}(C^{(t)}) = \max_{i=1}^{d+1} e_i^{(t)}$. We show the following invariant: (α) $\vec{0} \in \text{conv}(C^{(t)})$; and (β) $\mathcal{E}(C^{(t)}) < \mathcal{E}(C^{(t-1)})$ for $t \geq 1$. The invariant implies that eventually an $\mathcal{M}(j)$-colorful choice is returned.

Before the first iteration, the invariant holds since $C^{(0)} = P_1$. Assume we are now in iteration t and the invariant holds for all previous iterations. Due to Lemmas 5 and 6, we have $\vec{0} \in \text{conv}(C^{(t)})$ and thus Property (α) holds. Because $C^{(t-1)}$ was not an $\mathcal{M}(j)$-colorful choice (otherwise the balancing-steps would not haven executed), $\mathcal{E}(C^{(t-1)}) \geq 1$. Since Q contains only light color classes, adding Q to $C^{(t-1)}$ does not increase the excess. At least one point in C from each color P_i with $e_i^{(t-1)} \geq 1$ is replaced by Q. Hence, $\mathcal{E}(C^{(t)}) < \mathcal{E}(C^{(t-1)})$. Finally, we show that there are always $\mathcal{D}(j+1)+1$ light color classes for the recursion. By induction, the recursively computed set Q is an $\mathcal{M}(j+1)$-colorful choice. As C is pruned to at most $\mathcal{D}(j)+1$ points at the end of the balancing-steps, there are at most $\left\lfloor \frac{\mathcal{D}(j)+1}{\mathcal{M}(j)-\mathcal{M}(j+1)} \right\rfloor$ heavy color classes. One can show that this is at most $\mathcal{D}(j) - (\mathcal{D}(j+1)+1)$ for $d = \Omega(1/\varepsilon^3)$. Since we assumed $\varepsilon = \Omega(d^{-1/3})$, we can always find $\mathcal{D}(j+1) + 1$ light colors.

We now analyze the running time. Each iteration of the balancing-steps reduces the excess by at least one until the desired approximation guarantee is reached. Thus, the total number of iterations is bounded by $\mathcal{D}(j) + 1 - \mathcal{M}(j) = O(d)$. Each iteration requires $O(d^4)$ time. The recursion stops when $d = O(1)$ or

$\mathcal{M}(j) \geq \mathcal{D}(j) + 1$. In the first case, a perfect colorful choice is computed in $O(1)$ time. In the second case, we spend $O(d^4)$ time since pruning P_1 with Lemma 4 already gives a valid approximation. We can bound the recursion depth j until the second base case is reached. Since $\mathcal{M}(j) \geq \varepsilon(1-\varepsilon)^{j/2}(d+1)$ and $3(1-\varepsilon)^j(d+1) \geq \mathcal{D}(j)+1$, we have $\mathcal{M}(j) \geq \mathcal{D}(j)+1$ for $j = O((1/\varepsilon)\log(1/\varepsilon))$ using that $-\log(1-\varepsilon) = \Omega(\varepsilon)$. Thus, the total running time is $d^{O((1/\varepsilon)\log(1/\varepsilon))}$. □

4 A Subexponential Exact Algorithm

Now, we consider the case that we have "many" color classes instead of "only" $d+1$: given $\Theta(d^2 \log d)$ color classes, our algorithm computes a perfect colorful choice in $d^{O(\log d)}$ time, improving the brute force $d^{O(d)}$ algorithm. The algorithm follows the structure of Miller and Sheehy's algorithm for computing approximate Tverberg partitions [7]. We repeatedly combine m-colorful choices (for some m) to one $\lceil m/2 \rceil$-colorful choice. Eventually, we obtain a perfect colorful choice.

Lemma 7 *Let $C_1, \ldots, C_{d+1} \subset \mathbb{R}^d$ be m-colorful choices s.t. $|C_i| \leq d+1$ and s.t. $\vec{0} \in \operatorname{conv}(C_i)$. Then, a $\lceil m/2 \rceil$-colorful choice containing the origin in its convex hull can be computed in $O(d^5)$ time.*

Proof. First, we prune each set C_i with Lemma 4. This requires $O(d^5)$ time. Next, for $i = 1, \ldots, d+1$, we partition the colorful choice C_i into two sets $C_{i,1}, C_{i,2}$ of equal size s.t. the points from each color class are distributed evenly among the two sets. For each partition $C_{i,1}, C_{i,2}$, we apply Lemma 6 to obtain two representatives $c_{i,1}$ and $c_{i,2}$ in $O(d^3)$ time. By Lemma 6, we have $c_{i,1} \in \operatorname{pos}(C_{i,1})$ and $c_{i,2} \in \operatorname{pos}(C_{i,2})$. Since $\vec{0} \in \operatorname{conv}(\{c_{i,1}, c_{i,2}\})$, both points lie on a line through the origin and thus $-c_{i,1} \in \operatorname{pos}(C_{i,2})$. The $d+1$ points $c_{1,1}, c_{2,1}, \ldots, c_{d+1,1}$ are linearly dependent, so there exists a nontrivial linear combination $\vec{0} = \alpha_1 c_{1,1} + \cdots + \alpha_{d+1} c_{d+1,1}$. For $i = 1, \ldots d+1$, we let the set C contain $C_{i,1}$ if $\alpha_i > 0$ (since $c_{i,1} \in \operatorname{pos}(C_{i,1})$) and $C_{i,2}$ if $\alpha_i < 0$ (since $-c_{i,1} \in \operatorname{pos}(C_{i,2})$). By construction, C contains the origin in its convex hull and exactly one of $C_{i,1}, C_{i,2}$, for $i = 1, \ldots, d+1$. Since all sets C_i are m-colorful choices, C is a $\lceil m/2 \rceil$-colorful choice. □

Proof of Theorem 3. Let A be an array of size $k = \Theta(\log d)$. We set $c_0 = d+1$ and $c_i = \lceil c_{i-1}/2 \rceil$, for $i = 1, \ldots, k-1$. The ith cell of A stores a collection of c_i-colorful choices, such that each color class appears in exactly one colorful choice in A. Initially, $A[0]$ contains all $\Theta(d^2 \log d)$ color classes. We repeat the following steps, until we have computed a perfect colorful choice: let i be the maximum index s.t. $A[i]$ contains some $d+1$ sets $C_1, \ldots, C_{d+1}$. We apply Lemma 7 to obtain one c_{i+1}-colorful choice C. Let C' be the set C pruned with Lemma 4. If C' is a perfect colorful choice, we return it. Otherwise, we add it to $A[i+1]$. Furthermore, we add all colors that were removed during the pruning to $A[0]$. As these colors do not appear anywhere else in A, the invariant is maintained. We claim that a combination of $d+1$ sets in $A[k]$ for $k = \lceil \log(d+1) \rceil + 1$ results in a perfect colorful choice. We have $c_j \leq \frac{d+1}{2^k} + 2$. Thus, sets in $A[\lceil \log(d+1) \rceil]$ are 3-colorful choices, sets in $A[\lceil \log(d+1) \rceil + 1] = A[k]$ are 2-colorful choices and the combination of $d+1$ sets in $A[k]$ gives a perfect colorful choice. It remains to show that we can always make progress. The array has $k = \Theta(\log d)$ levels and a colorful choice has at most d colors. Thus, for $d^2 k + 1 = \Theta(d^2 \log d)$ colors, the pigeonhole principle implies that there is a cell with $d+1$ sets.

Let us consider the running time. One combination step takes $O(d^5)$ time. To compute a set in level i, we have to compute $d+1$ sets in level $i-1$. Hence, computing one set in level $k+1$ takes $d^{O(\log d)}$ time. □

Acknowledgements. We would like to extend our thanks to Frédéric Meunier and Pauline Sarrabezolles for interesting discussions on the colorful Carathéodory problem and for hosting us during a research stay at the École Nationale des Ponts et Chaussées.

References

[1] I. Bárány. A generalization of Carathéodory's theorem. *Discrete Math.*, 40(2–3):141–152, 1982.

[2] I. Bárány and S. Onn. Colourful linear programming and its relatives. *Math. Oper. Res.*, 22(3):550–567, 1997.

[3] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. System Sci.*, 37(1):79–100, 1988.

[4] J. Matoušek. *Lectures on discrete geometry*. Springer, 2002.

[5] F. Meunier and A. Deza. A further generalization of the colourful Carathéodory theorem. In *Discrete geometry and optimization*, volume 69 of *Fields Inst. Commun.*, pages 179–190. Springer, New York, 2013.

[6] F. Meunier and P. Sarrabezolles. Colorful linear programming, Nash equilibrium, and pivots. *arxiv:1409.3436*, 2014.

[7] G. L. Miller and D. R. Sheehy. Approximate centerpoints with proofs. *Comput. Geom.*, 43(8):647–654, 2010.

[8] W. Mulzer and Y. Stein. Complexity of finding nearest colorful polytopes. *Proceedings of the 30th European Workshop on Computational Geometry*, 2014.

[9] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.*, 48(3):498–532, 1994.

[10] K. S. Sarkaria. Tverberg's theorem via number fields. *Israel J. Math.*, 79(2–3):317–320, 1992.

[11] H. Tverberg. Further generalization of Radon's theorem. *J. London Math. Soc.*, 43:352–354, 1968.

Packing Segments in a Simple Polygon is APX-hard

Heuna Kim * Tillmann Miltzow †

Abstract For a given set of line segments and a polygon P in the plane, we want to find the maximum number of segments that can be disjointly embedded by translation into P. We show APX-hardness and discuss variations.

This problem can be considered in two respects : as a variant of the Kakeya problem and as a maximum-packing problem for line segments.

1 Introduction

The Kakeya Problem. The famous Kakeya problem asks for the region R in the plane with minimum-area such that a unit-length line segment can continuously rotate by π within R. One variant of the Kakeya problem relaxes the continuous rotation and tries to find a planar region R' with the minimum area such that translates of all the unit-length line segments in the plane can be placed in R'. The segments may intersect. This region R' is called a *minimum area translation cover*.

Pál [5, 4] solved these two problems, and many other interesting variations about the minimum-area translation cover have been studied (refer [3, 6] for surveys).

A Minimum-Container Problem and a 3-approximation Algorithm. Finding a minimum-area translation cover can be considered as a minimum-container problem if we want to *disjointly* embed line segments. The following question arises naturally in this context; given a set of line segments $\mathcal{S}$, what is the minimum-area convex body R such that translates of segments in $\mathcal{S}$ can be disjointly embedded in R?

We suspect this problem is computationally intractable, but not much is known about this problem except for a 3-approximation algorithm by Sang Won Bae (by private communication).

The 3-approximation algorithm is as follows. Using the algorithm by Ahn et al. [1], we compute the triangle T which is the minimum-area convex translation cover of the given set of line segments $\mathcal{S}$. Then, we construct a convex trapezoid Q as follows. First translate two copies T_1, T_2 of T so that one side of each copy is aligned on a line and T_1 and T_2 share one vertex v. We obtain the third copy T_3 by rotating T by π and translate it so that the three copies form the trapezoid $Q = T_1 \cup T_2 \cup T_3$, see Figure 1.

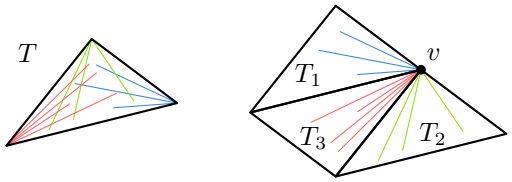


Figure 1: The minimum-area convex translation cover T and the trapezoid $Q = T_1 \cup T_2 \cup T_3$.

Then all segments in $\mathcal{S}$ can be disjointly embedded in Q; every segment s in $\mathcal{S}$ can be translated in a way that one of its endpoints lies on v and s still lies inside Q. Since the optimal area is at least the area of T, the obtained trapezoid gives a 3-approximation.

Problem Definition and Summary of Results. To solve a minimum-container problem it is natural to consider its dual, that is, a maximum-packing problem. We consider the maximum-packing problem in this abstract. We show hardness results for simple polygons and a simple approximation algorithm for convex polygons.

As in [2], we define MaxSegPack$d$ for a class $\mathcal{R}$ of regions in $\mathbb{R}^d$ as the following problem; given a collection of (open) segments and a region $R \in \mathcal{R}$, what is the maximum number of segments that can be disjointly embedded in R by translation?

This problem is known to be NP-hard when $\mathcal{R}$ is a convex 3-polytope of general regions in the plane [2]. We state the result for a convex 3-polytope.

Theorem 1 ([2]) MaxSegPack3 *for a convex 3-polytope is NP-hard.*

We state the main results as the following theorem.

Theorem 2 MaxSegPack2 *for a simple polygon and a set of unit segments $\mathcal{U}$ is strongly NP-complete. Also, approximating an optimal solution of* MaxSegPack2 *for a simple polygon and a set of unit segments with an approximation ratio $15/16 + \varepsilon$ is NP-hard for any $\varepsilon > 0$.*

We could also find a simple approximation algorithm.

*Institut für Informatik, Freie Universität Berlin, heunak@mi.fu-berlin.de. This research was supported by the Deutsche Forschungsgemeinschaft within the research training group Methods for Discrete Structures (GRK 1408).

†Institut für Informatik, Freie Universität Berlin, t.m@fu-berlin.de

Theorem 3 *There exists a k-approximation algorithm for* MaxSegPack2 *for a convex k-gon.*

By inspecting the proof from Theorem 1 in [2], we could easily conclude NP-hardness for high-dimensional cases.

We extend MaxSegPack$d$ to the following problem MaxPack(d_K, d_S) ; given a collection of (open) d_S-simplices and region R in d_K-space, what is the maximum number of simplices that can be disjointly embedded in R by translation?

Theorem 4 MaxPack(d_K, d_S) *for a convex d_K-polytope is NP-hard for all $d_K \geq 3, d_S \geq 1$.*

Remark. When a line segment s can be embedded in some region R, we say s *fits* in R. Also, if a set of line segments $\mathcal{S}$ can be disjointly embedded in R, we say $\mathcal{S}$ *can be packed* in R.

We regard two line segments of the same lengths and the same slopes as the same line segment since if two line segments have the same lengths and the same slopes we can overlap them completely by translation.

2 Proof of Theorem 2

We first show that MaxSegPack2 for a simple polygon P is in NP and then show that it is NP-hard. A natural candidate for a certificate of this problem is the set of the coordinates of the endpoints of the line segments. We can check whether the line segments are inside a given simple polygon P and whether they have no intersections by using linear inequalities.

We claim that those coordinates and the coefficients of linear inequalities can be described with polynomial precision. To this end, it is enough to show that the coordinates correspond to a feasible solution of conjunctions and disjunctions of a polynomial number of linear inequalities with coefficients of bounded precision.

To specify the linear inequalities, we first triangulate the given simple polygon arbitrarily. Three inequalities suffice to describe if each endpoint lies in one of the triangles. This gives us $6n$ inequalities, where n specifies the number of line segments we want to pack. A pair of line segments is crossing free if and only if at least one of them is completely to the left or completely to the right of the supporting line of the other. Since two linear inequalities suffice to describe if a line segment is to the left of another, this gives us $2\binom{n}{2}$ linear inequalities. Lastly, we need to specify two equalities per line segment to define the slope and the length of line segments (relative positions of two endpoints). In total, this gives us $6n + 2\binom{n}{2}$ inequalities and $2n$ equalities with coefficients of bounded precision. Hence, we can verify any certificate in a polynomial time.

Before describing the reduction from MAX-3-SAT, we state the following two lemmas for constructing gadgets. Lemma 5 will be used for the clause gadgets and Lemma 6 for the variable gadgets.

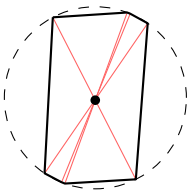


Figure 2: Four segments and a polygon such that exactly one of the segments fits but no two of them can be packed.

Lemma 5 *Let S be a set of unit-length line segments with distinct slopes. We construct a convex polygon $Q = Q(S)$ with the following properties:*

1. *any segment $s \in S$ fits in Q;*
2. *no two segments in S can be packed in Q; and*
3. *no unit-length line segment $s \notin S$ fits in Q.*

Proof. Translate all the segments of S so that their midpoints lie at the origin. Now define $Q(S)$ as the convex hull of all those segments; see Figure 2 for an illustration.

The diameter of Q is 1 and the diameter is attained only for pairs of opposite extreme points of Q. Therefore, a unit-length line segment s fits in Q if and only if s can be translated in a way that its endpoints lie at opposite extreme points of Q. This implies the first and the third property.

Each segment s that fits in Q has a unique position in Q and this unique position always goes through the origin. Thus, no two segments of unit length can be packed in Q. This implies the second property. □

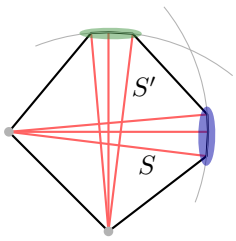


Figure 3: Sets S and S' and the convex polygon $R(S, S')$ constructed from them.

Lemma 6 *Let S be a set of unit length line segments such that the angle with the x-axis is within ± 0.1 radian, and let S' be a set of unit-length line segments such that the angle with the y-axis is within ± 0.1 radiant.*

There exists a convex polygon $R = R(S, S')$ with the following properties:

1. *segments in S can be packed in R;*
2. *the set S' can be packed in R;*
3. *no two segments $s \in S$ and $s' \in S'$ can be packed in R; and*
4. *no unit segment $s \notin S \cup S'$ fits into R.*

Proof. Translate the left endpoint of every line segment $s \in S$ to the point $(-0.5, 0)$ and the bottom endpoint of every line segment $s' \in S'$ to the point $(0, -0.5)$. The convex hull of those segments define $R = R(S, S')$. See Figure 3.

The diameter of Q is 1 and the diameter is attained only for pairs of points (p, q) such that either 1) $p = (-0.5, 0)$ and q is one of right extreme points (marked blue in Figure 3). or 2) $p = (0, -0.5)$ and q is one of top extreme points (marked green in Figure 3). These are exactly the endpoints of segments in $S \cup S'$ after we moved the segments of S. By the same argument as in Lemma 5, any unit-length line segment s fits in R if and only if $s \in S \cup S'$. Each segment s that fits in R has a unique position $p(s)$ in R. Observe that $p(s_1)$ and $p(s_2)$ are disjoint if either $s_1, s_2 \in S$ or $s_1, s_2 \in S'$ and $p(s_1)$ and $p(s_2)$ intersect otherwise. Thus, any two segments s_1 and s_2 can be packed in R if and only if either $s_1, s_2 \in S$ or $s_1, s_2 \in S'$. Altogether these arguments imply the above four properties. □

Given a 3-CNF formula ϕ with m clauses and n variables, we construct a simple polygon P and a set of $2m$ unit segments $\mathcal{U}$ that satisfy the following property; there exists an assignments that satisfies t clauses of ϕ if and only if $t + m$ elements of $\mathcal{U}$ can be disjointly embedded in P.

We begin by defining the line segments. Then we describe clause and variable polygons and finally we describe how to join everything to one big polygon.

For each clause C_i, $i = 1, \ldots, m$ of ϕ we construct two unit segments s_i and s'_i. The line segment s_i forms an angle $\alpha_i = \frac{i}{100m}$ with the x-axis and s'_i forms an angle $\alpha'_i = \frac{i}{100m}$ with the y-axis. [1] Note that all s_i's can be regarded as slight perturbations of a horizontal unit segment, and all s'_i as a slight perturbation of a vertical unit segment.

For each clause C_i we define the *clause polygon*

$$Q_i = Q(\{s_i, s'_i\})$$

according to Lemma 5.

For each variable x_j with $j = 1, \ldots, n$, we define

$$S_j = \{\, s_i \mid \text{the literal } x_j \text{ is contained in } C_i \,\} \text{ and}$$

$$S'_j = \{\, s'_i \mid \text{the literal } \overline{x_j} \text{ is contained in } C_i \,\}.$$

[1] To compute the endpoints of the segments we need sine and cosine operations, but it is not necessary since the construction does not depend on the exact values of the angles. We also could define the angles as rational values.

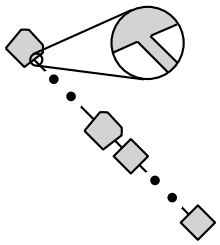


Figure 4: Joining polygons together without new segments fitting in.

For each variable x_j we define the *variable polygon*

$$R_j = R(S_j, S'_j)$$

according to Lemma 6. Note that each segment $s \in \mathcal{U}$ fits in at most four polygons: one clause polygon and at most three variable polygons.

The polygon P is defined by joining all the polygons $Q_1, \ldots, Q_m, R_1, \ldots, R_n$. In order to join these polygons, add a narrow diagonal tunnel from one polygon to the next; see Figure 4 for an illustration. Since every segment in $\mathcal{U}$ is either almost horizontal or vertical, none of them fits into the tunnel.

It is clear that this construction can be done within a polynomial time. For this polygon P and this set of line segments $\mathcal{U}$, we claim that there exists an assignment that satisfies t clauses of ϕ if and only if $t + m$ elements of $\mathcal{U}$ can be packed in P.

First suppose that we are given an assignment A that satisfies t clauses of ϕ. We will describe how to embed $t + m$ segments in the polygon P. There are some segments that fit in P not uniquely but in several possible variable polygons. In this case, we make an arbitrary choice. If x_j is true in A, place segments in S_j in the variable polygon R_j and if x_j is false in A, place segments in S'_j in R_j unless the segments are already placed in some other variable polygon. We also place all remaining segments into their corresponding clause polygon Q_i if possible.

If C_i is satisfied by A, both segments s_i and s'_i are placed in P for the following reason. Either s_i or s'_i is placed in R_j for some j since at least one variable x_j in C_i makes C_i satisfied. We placed the other to Q_i unless it is already contained in a different variable polygon.

Otherwise, only one of the segments s_i or s'_i fits in P, since neither s_i nor s'_i are contained in any variable polygon R_j and both segments cannot fit in Q_i. Since t clauses are satisfied, the first case happens t times and the second case appears $m - t$ times. Hence, $t + m$ segments can be packed into P.

For the other direction, suppose $t + m$ segments in $\mathcal{U}$ can be packed in P. We assume this packing is maximal. We define an assignment A by checking which segments are placed in R_j. If R_j contains a segment of S_j then we set x_j to true and otherwise

we set x_j to false. We can repeat the same argument in the other direction. For each clause C_i, if s_i and s'_i are both packed, then either s_i or s'_i is in some R_j, which implies that the clause C_i is satisfied by the variable x_j. Otherwise, one of s_i and s'_i is packed, but none of s_i and s'_i is placed in a variable polygon, and this implies that C_i cannot be satisfied by A. Then $2 \times n_S + n_N = t + m$ and $n_S + n_N = m$ where n_S is the number of satisfied clauses and n_N is the number of non-satisfied clauses. Then the number of satisfied clause in A is t. This shows the problem is NP-hard.

Finally we show that no approximation algorithm exists with an approximation ratio $15/16 + \varepsilon$ for any $\varepsilon > 0$. Suppose there exists an approximation algorithm for MaxSegPack2 for a simple polygon and a set of unit length segments with an approximation ratio $15/16+\varepsilon/2$ for some $\varepsilon > 0$. By using the previous construction for any CNF formula ϕ of m clauses, we can find an assignment A that satisfies t clauses where $\frac{t+m}{2m} \geq 15/16+\varepsilon/2$; that is, we have an approximation algorithm for MAX-3-SAT with an approximation ratio $t/m \geq 7/8 + \varepsilon$.

Since there is no approximation algorithm for MAX-3-SAT with the approximation ratio $7/8 + \varepsilon$ for any $\varepsilon > 0$ unless P=NP, there exists no approximation algorithm for MaxSegPack2 for a simple polygon and a set of unit segments with an approximation ratio $15/16+\varepsilon/2$ for any $\varepsilon/2 > 0$ unless P=NP.

3 Approximation Algorithm for a Convex k-gon

The following algorithm gives a k-approximation for MaxSegPack2 for a convex polygon.

Input: a set of line segments $\mathcal{S}$; convex k-gon P
Output: $\mathcal{T} \subseteq \mathcal{S}$; a k-approximated solution
for all $v \in$ vertices of P **do**
  $S_v := \{s \in \mathcal{S} : s$ can be placed on v inside $P\}$
end for
return the largest set S_v

Any segment $s \in \mathcal{S}$ that fits in P can be translated so that one of endpoints of v is on a vertex of P and v still lies in P. For each vertex v of P, all the elements S_v can be packed in P. Since

$$\bigcup_{p:\text{vertices of } P} S_v$$

is at least the optimal solution, the largest set S_v has the cardinality at least $1/k$ of the optimal solution.

4 Hardness for d-space

Theorem 1 in [2] states MaxSegPack3 for a convex 3-polytope is NP-hard; that is, MaxPack$(3, 1)$ is NP-hard. In the proof, all line segments were constructed in a way that they are uniquely embeddable in a convex 3-polytope for the reduction. We can prove

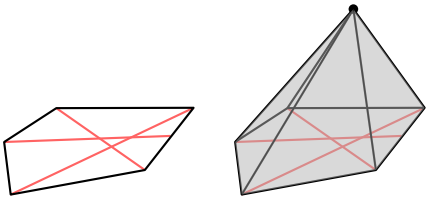


Figure 5: Visualization of constructing a pyramid, in dimension three.

that MaxPack(d_K, d_S) for a convex d_K-polytope is NP-hard inductively by reducing (1) an instance of MaxPack$(d_K, 1)$ to an instance of MaxPack$(d_K + 1, 1)$ and (2) an instance of MaxPack(d_K, d_S) to an instance MaxPack$(d_K + 1, d_S + 1)$.

Let $(K, \mathcal{S})$ be any instance of MaxPack$(d_K, 1)$ where K is a convex d_K-polytope and $\mathcal{S}$ a set of line segments that can be uniquely embedded in K. We construct K' by taking a pyramid whose base is K. Then K' is convex $(d_K + 1)$-polytope. Then, $(K', \mathcal{S})$ is an instance of MaxPack$(d_K + 1, 1)$ whose solution corresponds to a solution of $(K, \mathcal{S})$ for MaxPack$(d_K, 1)$, since all line segments $s \in \mathcal{S}$ can be embedded in K uniquely and s cannot be embedded in any smaller homothetic copies of K. This is the reduction for (1), and the reduction for (2) is quite similar; we replace $s \in \mathcal{S}$ by the convex hull s' of s and the apex of K'. Therefore, MaxPack(d_K, d_S) is NP-hard for all $d_K \geq 3, d_S \geq 1$.

Acknowledgments We would like to thank Paul Seiferth and Yannik Stein for helpful discussions and proofreading. We also thank Sang Won Bae for communicating the 3-approximation algorithm and Günter Rote for the argument that MaxSegPack2 for a simple polygon is in NP.

References

[1] H.-K. Ahn, S. W. Bae, O. Cheong, J. Gudmundsson, T. Tokuyama, and A. Vigneron. A generalization of the convex Kakeya problem. In *LATIN 2012: Theoretical Informatics*, pages 1–12. Springer, 2012.

[2] M. G. Dobbins and H. Kim. Packing segments in a convex 3-polytope is np-hard. *30th European Workshop on Computational Geometry (EuroCG)*, 2014.

[3] I. Łaba. From harmonic analysis to arithmetic combinatorics. *Bulletin (New Series) of the American Mathematical Society*, 45(1):77–115, 2008.

[4] J. Pál. Ein Minimumproblem für Ovale. *Mathematische Annalen*, 83(3):311–319, 1921.

[5] J. Pál. *Ueber ein elementares Variationsproblem*, volume 3. AF Host, 1921.

[6] T. Tao. From rotating needles to stability of waves: Emerging connections between. *Notices of the AMS*, 48(3), 2001.

Finding Pairwise Intersections Inside a Query Rectangle

Mark de Berg* Ali D. Mehrabi*

Abstract

We study the following problem: preprocess a set $\mathcal{O}$ of objects in the plane into a data structure that allows us to efficiently report all pairs of objects from $\mathcal{O}$ that intersect inside an axis-aligned rectangular query range Q. We present data structures of size $O(n \operatorname{polylog} n)$ and with query time $O((k+1)\operatorname{polylog} n)$ time, where k is the number of reported pairs, for two classes of objects: axis-aligned rectangles and objects with small union complexity. For axis-aligned rectangles we obtain a data structure with $O(n \log n)$ storage and $O((k+1)\log n \log^* n)$ query time, where k is the number of reported pairs of objects. Our data structure for objects with small union complexity—disks and other classes of fat objects are examples—uses $O(U(n)\log n)$ storage, where $U(n)$ is maximum complexity of the union of n objects from the given class, and it has $O((k+1)\log^2 n)$ query time.

We assume throughout the paper that the objects in $\mathcal{O}$ as well as the query rectangles are closed sets.

1 Introduction

The study of geometric data structures is an important subarea within computational geometry, and range queries form one of the most widely studied topics within this area. In a range query, the goal is to report or count all points from a given set $\mathcal{O}$ that lie inside a query range Q. The more general version, where $\mathcal{O}$ contains other objects than just points and the goal is to report all objects intersecting Q, is often called intersection searching and it has been studied extensively as well.

A common characteristic of the range-searching and intersection-searching problems studied so far, is that whether an object $o_i \in \mathcal{O}$ should be reported (or counted) depends only on o_i and Q. In this paper we study a range-searching variant where we are interested in reporting *pairs* of objects that satisfy a certain criterion. In particular, we want to preprocess a set $\mathcal{O} = \{o_1, \ldots, o_n\}$ of n objects in the plane such that, given a query range Q, we can efficiently report all pairs of objects o_i, o_j that intersect inside Q. An obvious approach is to precompute all intersections between the objects and store the intersections in a suitable intersection-searching data structure. This may give fast query times, but in the worst case any two objects intersect, so $\Omega(n^2)$ is a lower bound on the storage for this approach. The main question is thus: can we achieve fast query times with a data structure that uses subquadratic (and preferably near-linear) storage in the worst case?

We answer this question affirmatively for the case where Q is an axis-aligned rectangle and for two classes of objects: axis-aligned rectangles and objects with small union complexity.

*Department of Computer Science, TU Eindhoven, the Netherlands. MdB and AM are supported by the Netherlands Organization for Scientific Research (NWO).

2 Axis-aligned segments and rectangles

In this section we study the case where the set $\mathcal{O}$ is a set of n rectangles in the plane. As a warm-up exercise we start with the case where $\mathcal{O}$ consists of axis-aligned segments. Our approach for these two cases is the same and uses the following two-step query process.

1. Compute a *seed set* $\mathcal{O}^*(Q) \subseteq \mathcal{O}$ of objects such that the following holds: for any two objects o_i, o_j in $\mathcal{O}$ such that o_i and o_j intersect inside Q, at least one of o_i, o_j is in $\mathcal{O}^*(Q)$.
2. For each seed object $o_i \in \mathcal{O}^*(Q)$, perform an intersection query with the range $o_i \cap Q$ in the set $\mathcal{O}$, to find all objects $o_j \neq o_i$ intersecting o_i inside Q.

To make this approach efficient, we require that the seed set $\mathcal{O}^*(Q)$ does not contain too many objects that do not give an answer in Step 2. More precisely, if k denotes the number of pairs of objects in $\mathcal{O}$ that intersect inside Q, then we require that $|\mathcal{O}^*(Q)| = O(k+1)$.

Axis-aligned segments. Let $\mathcal{O} = \{s_1, \ldots, s_n\}$ be a set of axis-aligned segments, and let $\mathsf{V}(\mathcal{O})$ and $\mathsf{H}(\mathcal{O})$ denote the set of vertical and horizontal segments in $\mathcal{O}$, respectively. We assume for simplicity that we are only interested in intersections between horizontal and vertical segments; the solution can easily be adapted to the case where we also want to report intersections between two horizontal (or two vertical) segments.

The key to our approach is to be able to efficiently find the seed set $\mathcal{O}^*(Q)$. To this end, during the preprocessing we compute an $O(n)$-sized subset W of the intersection points in $\mathcal{O}$. We call intersection points in W *witnesses*. The witness set W is defined as follows: for each line segment $s_i \in \mathsf{V}(\mathcal{O})$ we put the

topmost and bottommost intersection points of s_i with a segment from $\mathsf{H}(\mathcal{O})$ (if any) into W; for each line segment $s_i \in \mathsf{H}(\mathcal{O})$ we put the leftmost and rightmost intersection points of s_i with a segment from $\mathsf{V}(\mathcal{O})$ (if any) into W. Since we take at most two witness points for each line segment, the size of W is clearly at most $2n$. Our data structure to find the seed set $\mathcal{O}^*(Q)$ now consists of three components.

- We store the witness set W in a data structure $\mathcal{D}_1$ for 2-dimensional orthogonal range reporting.
- We store $\mathsf{V}(\mathcal{O})$ in a data structure $\mathcal{D}_2$ that allows us to decide if there are any segments that completely cross the query rectangle Q from top to bottom. The data structure should also be able to report all such segments.
- We store $\mathsf{H}(\mathcal{O})$ in a data structure $\mathcal{D}_3$ that allows us to decide if there are any segments that completely cross the query rectangle Q from left to right.

Step 1 of our query process, where we find the seed set $\mathcal{O}^*(Q)$, now proceeds as follows.

1(i) Perform a query in $\mathcal{D}_1$ to find all witness points inside Q. For each reported witness point, insert the corresponding segment into $\mathcal{O}^*(Q)$.

1(ii) Perform queries in $\mathcal{D}_2$ and $\mathcal{D}_3$ to decide if the number of segments crosses Q completely from top to bottom, and the number of segments crosses Q completely from left to right, are both non-zero. If so, report all segments crossing completely from top to bottom, and put them into $\mathcal{O}^*(Q)$.

Lemma 1 *Let s_i, s_j be two segments in $\mathcal{O}$ such that $s_i \cap s_j \in Q$. Then at least one of s_i, s_j is put into $\mathcal{O}^*(Q)$.*

Proof. If s_i crosses Q completely from left to right and s_j crosses Q completely from top to bottom (or vice versa), then one of them will be put into $\mathcal{O}^*(Q)$ in Step 1(ii). Otherwise at least one of the segments, say s_i, has an endpoint v inside Q. But then the intersection point on s_i closest to v, which is a witness point, must lie inside Q. Hence, s_i is put into $\mathcal{O}^*(Q)$ in Step 1(i). □

Recall that in Step 2 of the query procedure we need to report, for each segment s_i in the seed set $\mathcal{O}^*(Q)$, the segments $s_j \in \mathcal{O}$ intersecting $s_i \cap Q$. Thus we need to store $\mathcal{O}$ in a data structure $\mathcal{D}_4$ that allows us to report all segments intersecting an axis-aligned query segment. Putting everything together we obtain the following theorem.

Theorem 2 *Let $\mathcal{O}$ be a set of n axis-aligned segments in the plane. Then there is a data structure that uses $O(n \log n)$ storage and that allows us to report, for any axis-aligned query rectangle Q, all pairs of segments s_i, s_j in $\mathcal{O}$ such that s_i intersects s_j inside Q in $O((k+1) \log n \log^* n)$ time, where k denotes the number of answers.*

Proof. For the data structure $\mathcal{D}_1$ on the set W we can take a standard 2-dimensional range tree [2], which uses $O(n \log n)$ storage. If we apply fractional cascading [2], reporting the witness points inside Q takes $O(\log n + k_w)$ time, where k_w is the number of reported witness points. For $\mathcal{D}_2$ (and, similarly, $\mathcal{D}_3$) we note that a vertical segment $s_i := x_i \times [y_i, y_i']$ crosses $Q := [x_Q, x_Q'] \times [y_Q, y_Q']$ if and only if the point (x_i, y_i, y_i') lies in the range $[x_Q, x_Q'] \times [-\infty, y_Q] \times [y_Q', \infty]$. Hence, we can use the data structure of Subramanian and Ramaswamy [9], which uses $O(n \log n)$ storage and has $O(\log n \log^* n + \#\text{answers})$ query time. Hence, the supporting data structures for Step 1 use $O(n \log n)$ storage, and finding the seed set takes $O(\log n \log^* n + |\mathcal{O}^*(Q)|)$ time.

It remains to analyze Step 2 of the query procedure. First notice that the problem of finding for a given $s_i \in \mathcal{O}^*(Q)$ all $s_j \in \mathcal{O}$ such that $s_i \cap Q$ intersects s_j, is the same range-searching problem as Step 1(ii), except that the query range is a line segment this time. Hence, we again transform the problem to a 3D range-searching problem and use the data structure of Subramanian and Ramaswamy [9]. Thus the running time of Step 2 is $\sum_{s_i \in \mathcal{O}^*(Q)} O(\log \log^* n + k_i)$, where k_i denotes the number of segments in $\mathcal{O}$ that intersect s_i inside Q. Since $|\mathcal{O}^*(Q)| \leqslant 2k$ where k is the total number of reported pairs—each segment in $\mathcal{O}^*(Q)$ intersects at least one other segment inside Q and for every reported pair we put at most two segments into the seed set—the time for Step 2 is $O(|\mathcal{O}^*(Q)| \log n \log^* n + k) = O((k+1) \log n \log^* n)$. □

Axis-aligned rectangles. Let $\mathcal{O} = \{r_1, \ldots, r_n\}$ be a set of axis-aligned rectangles in the plane. As before, we first define a witness set W. The witnesses in W are now axis-aligned segments rather than just points. For each rectangle $r_i \in \mathcal{O}$ we define at most ten witness segments, two for each edge of r_i and two in the interior of r_i, as follows; see Fig. 1. Let e be an edge of r_i, and consider the set $S(e) := e \cap \left(\bigcup_{j \neq i} r_j\right)$, that is, the part of e covered by the other rectangles. The set $S(e)$ consists of a number of sub-edges of e. If e is vertical then we add the topmost and bottommost sub-edge from $S(e)$ (if any) to W; if e is horizontal we add the leftmost and rightmost sub-edge to W. The two witness segments in the interior of r_i are defined as follows. Suppose that there are vertical edges (belonging to other rectangles r_j) that completely cross r_i from top to bottom. Then we put $e' \cap r_i$ into W, where e' is the rightmost such crossing edge. Similarly, we put into W the topmost horizontal edge e'' that completely crosses r_i from left to right.

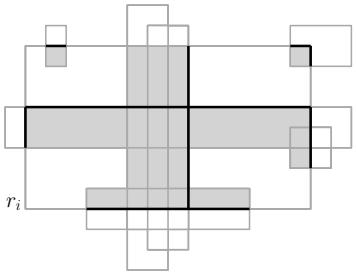


Figure 1: The witness segments for a rectangle (r_i). The gray areas indicate all intersections with r_i. The black segments form the set of all witness segments of r_i.

Our data structure to find the seed set $\mathcal{O}^*(Q)$ now consists of the following components.

- We store the witness set W in a data structure $\mathcal{D}_1$ that allows us to report the set of segments that intersect the query rectangle Q.
- We store the vertical edges of the rectangles in $\mathcal{O}$ in a data structure $\mathcal{D}_2$ that allows us to decide if the set $\mathsf{V}(Q)$ of edges that completely cross a query rectangle Q from top to bottom, is non-empty. The data structure should also be able to report all (rectangles corresponding to) the edges in $\mathsf{V}(Q)$.
- We store the horizontal edges of the rectangles in $\mathcal{O}$ in a data structure $\mathcal{D}_3$ that allows us to decide if the set $\mathsf{H}(Q)$ of edges that completely cross a query rectangle Q from left to right, is non-empty.
- We store $\mathcal{O}$ in a data structure $\mathcal{D}_4$ that allows us to report the set of rectangles that contain a query point q.

Step 1 of our query process, where we find the seed set $\mathcal{O}^*(Q)$, now proceeds as follows.

1(i) Perform a query in $\mathcal{D}_1$ to find all witness segments intersecting Q. For each reported witness segment, insert the corresponding rectangle into $\mathcal{O}^*(Q)$.

1(ii) Perform queries in $\mathcal{D}_2$ and $\mathcal{D}_3$ to decide if the sets $\mathsf{V}(Q)$ and $\mathsf{H}(Q)$ are both non-empty. If so, report all rectangles corresponding to edges in $\mathsf{V}(Q)$ and put them into $\mathcal{O}^*(Q)$.

1(iii) For each corner point q of Q, perform a query in $\mathcal{D}_4$ to report all rectangles in $\mathcal{O}$ that contain q, and put them into $\mathcal{O}^*(Q)$.

The next lemma proves the correctness of this procedure. We prove the lemma in the full version using a case analysis.

Lemma 3 *Let r_i, r_j be two rectangles in $\mathcal{O}$ such that $(r_i \cap r_j) \cap Q \neq \emptyset$. Then at least one of r_i, r_j is put into $\mathcal{O}^*(Q)$.*

In the second part of the query procedure we need to report, for each rectangle r_i in the seed set $\mathcal{O}^*(Q)$, the rectangles $r_j \in \mathcal{O}$ intersecting $r_i \cap Q$. Thus we need to store $\mathcal{O}$ in a data structure $\mathcal{D}_5$ that allows us to report all rectangles intersecting an axis-aligned query rectangle. Putting everything together, and plugging in standard data structures as before, we obtain the following theorem.

Theorem 4 *Let $\mathcal{O}$ be a set of n axis-aligned rectangles in the plane. Then there is a data structure that uses $O(n \log n)$ storage and that allows us to report, for any axis-aligned query rectangle Q, all pairs of rectangles r_i, r_j in $\mathcal{O}$ such that r_i intersects r_j inside Q in $O((k+1) \log n \log^* n)$ time, where k denotes the number of answers.*

3 Objects with small union complexity

In the previous section we presented efficient solutions for the case where $\mathcal{O}$ consists of axis-aligned rectangles. In this section we obtain results for classes of constant-complexity objects with small union complexity. More precisely, we need that $U(n)$, the maximum union complexity of any set of n objects from the class, is small. This is for instance the case for disks (where $U(n) = O(n)$ [8]) and for locally fat objects (where $U(n) = O(n2^{O(\log^* n)})$ [1]).

Recall that in Step 2 of the query algorithm of the previous section, we needed to perform a range query with $o_i \cap Q$ for each $o_i \in \mathcal{O}^*(Q)$. When we are dealing with curved objects, this will be an expensive query. Hence, to deal with curved objects we modify our query procedure.

1. Compute a seed set $\mathcal{O}^*(Q) \subseteq \mathcal{O}$ of objects such that the following holds: for any two objects o_i, o_j in $\mathcal{O}$ such that o_i and o_j intersect inside Q, both o_i and o_j are in $\mathcal{O}^*(Q)$.
2. Compute all intersecting pairs of objects in the set $\{o_i \cap Q : o_i \in \mathcal{O}^*(Q)\}$ by a plane-sweep algorithm.

The main question is now how to efficiently find $\mathcal{O}^*(Q)$, which should contain all objects intersecting at least one other object inside Q. Next we describe how to do this when the union complexity $U(m)$ is small. For each object $o_i \in \mathcal{O}$ we define $o_i^* := \bigcup_{o_j \in \mathcal{O}, j \neq i} (o_i \cap o_j)$ as the union of all intersections between o_i and all other objects in $\mathcal{O}$. Let $|o_i^*|$ denote the complexity (that is, number of vertices and edges) of o_i^*.

Lemma 5 $\sum_{i=1}^{n} |o_i^*| = O(U(n))$.

Proof. Consider the arrangement induced by the objects in $\mathcal{O}$. We define the *level* of a vertex v in this arrangement as the number of objects from $\mathcal{O}$ that contain v in their interior. We claim that every vertex of any o_i^* is a level-0 or level-1 vertex. Indeed, a level-k vertex for $k > 1$ is covered by k objects and therefore it will be in interior of the intersection of those k objects.

This is easily seen to imply that it cannot be a vertex of any o_i^*.

Since the level-0 vertices are exactly the vertices of the union of $\mathcal{O}$, the total number of level-0 vertices is $U(n)$. Moreover, it follows from the Clarkson-Shor technique [5] that the total number of level-1 vertices is $O(U(n))$ as well. The lemma now follows from the fact that each level-0 or level-1 vertex contributes to at most two different o_i^*'s. □

Our goal in Step 1 is to find all objects o_i such that o_i^* intersects Q. To this end consider the connected components of o_i^*. If o_i^* intersects Q then one of these components lies completely inside Q or an edge of Q intersects o_i^*. By taking a representative point inside every component of each o_i^* and storing these points in a structure for orthogonal range searching, we can find the components that are completely inside Q, leading to the following lemma.

Lemma 6 *We can find all o_i^* that have a component completely inside Q in $O(\log n + k)$ time, where k is the number of pairs of objects that intersect inside Q, with a data structure that uses $O(U(n)\log n)$ storage.*

Next we describe a data structure for reporting all o_i^* intersecting a vertical edge of Q; finding the o_i^* intersecting a horizontal edge can be done in a similar way.

Our data structure is a balanced binary tree $\mathcal{T}$, whose leaves in $\mathcal{T}$ are in one-to-one correspondence to the objects in $\mathcal{O}$. For an (internal or leaf) node ν in $\mathcal{T}$, let $\mathcal{T}(\nu)$ denote the subtree rooted at ν and let $\mathcal{O}(\nu)$ denote the set of objects corresponding to the leaves of $\mathcal{T}(\nu)$. Define $\mathcal{U}(\nu) := \bigcup_{o_i \in \mathcal{O}(\nu)} o_i^*$. At node ν, we store a point-location data structure on the trapezoidal map of $\mathcal{U}(\nu)$. (If the objects are curved, then the "trapezoids" in the map actually have curved top and bottom edges.)

Lemma 7 *The tree $\mathcal{T}$ uses $O(U(n)\log n)$ storage and allows us to report all o_i^* intersecting a vertical edge s of Q in $O((t+1)\log^2 n)$ time, where t is the number of answers.*

Proof. To report all o_i^* intersecting s we walk down $\mathcal{T}$, only visiting those nodes ν such that s intersects $\mathcal{U}(\nu)$. This way we end up in the leaves corresponding to the o_i^* intersecting s. To decide if we have to visit a child ν of an already visited node, we do a point location with both endpoints of s in the trapezoidal map of $\mathcal{U}(\nu)$. Now s intersects $\mathcal{U}(\nu)$ if and only if one of these endpoints lies in a trapezoid inside $\mathcal{U}(\nu)$ and/or the two endpoints lie in different trapezoids. Thus we spend $O(\log n)$ time to decide if we have to visit a node. Since we visit $O(k \log n)$ nodes, the total query time is as claimed.

To analyze the storage we claim that the sum of the complexities of $\mathcal{U}(\nu)$ over all nodes ν at any fixed height of $\mathcal{T}$ is $O(U(n))$. The bound on the storage then follows from the fact that the point location data structures on the trapezoidal maps take linear space [6], and the fact that the height of $\mathcal{T}$ is $O(\log n)$. It remains to prove the claim. Consider a node ν at a given height h in $\mathcal{T}$. It can be shown that each vertex in $\mathcal{U}(\nu)$ is either a level-0 or level-1 vertex of the arrangement induced by the objects in $\mathcal{O}(\nu)$, or a vertex of o_i^*, for some o_i in $\mathcal{O}(\nu)$. The number of vertices of the former type is $O(U(|\mathcal{O}(\nu)|))$, which sums to $O(U(n))$ over all nodes at height h. By Lemma 5 the number of vertices of the latter type over all nodes at height h sums to $O(U(n))$. □

Theorem 8 *Let $\mathcal{O}$ be a set of n constant-complexity objects in the plane from a class of objects such that the maximum union complexity of any m objects from the class is $U(m)$. Then there is a data structure that uses $O(U(n)\log n)$ storage and that allows us to report for any axis-aligned query rectangle Q, in $O((k+1)\log^2 n)$ time all pairs of objects o_i, o_j in $\mathcal{O}$ such that o_i intersects o_j inside Q, where k denotes the number of answers.*

References

[1] B. Aronov, M. de. Berg, E. Ezra, and M. Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM J. Comput.* 43(2):543–572 (2014).

[2] M. de. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications (3rd edition).* Springer-Verlag, 2008.

[3] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.* 15:703–724 (1986).

[4] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.* 17:427–462 (1988).

[5] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discr. Comput. Geom.* 4:387–421 (1989).

[6] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.* 15:317-340 (1986).

[7] H. Edelsbrunner, M. H. Overmars, and R. Seidel. Some methods of computational geometry applied to computer graphics. *Comput. Vision, Graphics and Image Proc.* 28:92–108 (1984).

[8] K. Keden, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discr. Comput. Geom.* 1:59-71 (1986).

[9] S. Subramanian, and S. Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proc. 6th ACM-SIAM Symp. Discr. Alg.*, pages 378–387, 1995.

Computing the Smallest Color-Spanning Equaliteral Triangle

Javad Hasheminejad* Payam Khanteimouri * Ali Mohades *

Abstract

Let $\mathcal{P}$ be a set of n colored points with k colors in the plane. A region is color-spanning if it contains at least one point from each color. In this paper, we study the problem of computing the smallest color-spanning equilateral triangle whose one side is parallel to x-axis. We first show that the number of the minimal color-spanning equilateral triangles is $O(n)$ in the worst case. Then, we present an efficient algorithm running in $O(n \log n)$ time to solve the problem. Finally, we show that our algorithm can be used to compute a 2-approximation of the smallest perimeter color-spanning convex hull which is the first subquadratic time algorithm with approximation factor 2.

1 Introduction

Background. In many practical problems in computational geometry, the input points are imprecise. An imprecise point can be defined by a region where the exact location of the point can be anywhere inside the region [7]. In a case that an imprecise point is modeled with a discrete range such as a point set, each imprecise point can be represented by a set of points which determines all possible locations of that point. Therefore, by assigning a distinct color to each imprecise point, a set of k imprecise points can be modeled by n points with k colors, where n is the number of possible locations for all imprecise points. In this point of view, the problem is to choose exactly k points with different colors in such a way that a geometric structure e.g. convex hull, diameter, bounding box, etc. gets minimized or maximized —see [5] and references therein. Furthermore, we have similar models in other areas such as facility location, statistical clustering, pattern recognition and generalized range searching [1, 3, 8].

Related works. Suppose we are given a set of n points with k colors in the plane. A region is said to be *color-spanning* if it contains at least one point from each color. Abellanas et al. [1] presented an algorithm to compute *the smallest color-spanning axis-parallel rectangle* in $O(n(n-k)\log^2 k)$ time. The problem of computing *the smallest color-spanning circle* can be solved in $O(nk \log n)$ time using the upper envelope of Voronoi surfaces [4]. In addition, Khanteimouri et al. [6] presented an $O(n \log^2 n)$ time algorithm to compute *the smallest color-spanning axis-parallel square*. Moreover, For the problem of computing *the smallest perimeter color-spanning convex hull*, Ju et al. [5] showed the NP-hardness of the problem, and they proposed a π-approximation algorithm running in $O(n^2+nk \log k)$ time. They also proposed a $\sqrt{2}$-approximation algorithm for the problem running in the same time of computing the smallest color-spanning rectangle.

Our Results. In this paper, we study the problem of computing the smallest color-spanning equilateral triangle whose one side is parallel to x-axis (SCST, for short). Beside the applications of this problem in location planning, the perimeter of the SCST approximates the perimeter of the smallest color-spanning convex hull. In Section 2, we first show that there are $O(n)$ minimal color-spanning triangles in the worst case. Then, we present an $O(n \log n)$ time algorithm to compute the SCST. Next in Section 3, we show that the perimeter of the SCST gives a 2-approximation for the perimeter of the smallest color-spanning convex hull which significantly improves the first result presented in [5]. In fact, no subquadratic time algorithm with approximation factor 2 was known before. Finally, we conclude in Section 4.

2 Computing the Smallest Color-Spanning Equilateral Triangle

In this section, we focus on the problem of computing the smallest color-spanning equilateral triangle whose base is parallel to the x-axis. We start by some preliminaries and definitions.

2.1 Preliminaries and Definitions

Let $\mathcal{S} = \{s_1, \cdots, s_n\}$ be a set of points in the plane.

- For a point $s_i \in \mathcal{S}$, *wedge* of s_i is the portion of the plane restricted between two half-lines with slopes $\pm\sqrt{3}$ starting from s_i towards increasing y-coordinates—see Figure 1-(a). We denote the wedge of s_i by $\mathcal{W}(s_i)$.
- We say s_i *dominates* s_j if and only if $\mathcal{W}(s_i) \subseteq \mathcal{W}(s_j)$. A point s_i is *maximal* if it is not dominated by any other point in $\mathcal{S}$.

*Department of Mathematics and Computer Science, Amirkabir University of Technology, Iran

- For a set of m functions $\mathcal{F} = \{f_1, \cdots, f_m\}$ we define $\mathcal{L}_\mathcal{F}$ as *the lower envelope* of $\mathcal{F}$. Precisely, $\mathcal{L}_\mathcal{F}(x) = \min_{f\in\mathcal{F}} f(x)$. Similarly, $\mathcal{U}_\mathcal{F} = \max_{f\in\mathcal{F}} f(x)$ denotes *the upper envelope* of $\mathcal{F}$.

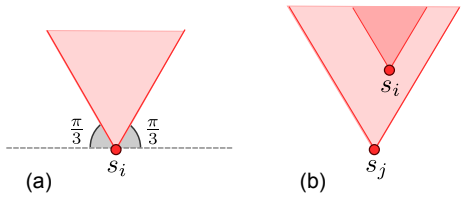


Figure 1: (a) The wedge of point s_i. (b) s_i dominates point s_j.

Now, let $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cdots \cup \mathcal{P}_k$ be the given set of n points with k colors in the plane where $\mathcal{P}_i$ is the set of points with color i.

- We define $\mathcal{L}_i$ as *the lower envelope of color* i which is the lower envelope of boundaries of $\mathcal{W}(p)$ for all $p \in \mathcal{P}_i$.
- $\mathcal{L}_i$ consists of *mountains* and *valleys*. In fact, a valley is a point in $\mathcal{P}_i$ that does not dominate any other point of $\mathcal{P}_i$ and a mountain is a local maximum point which appears between two consecutive valleys. In addition, we define two additional mountains at $-\infty$ and $+\infty$ for each $\mathcal{L}_i$—see Figure 2 for more illustration.

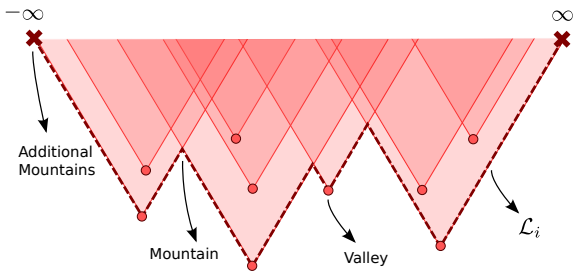


Figure 2: The mountains and valleys on $\mathcal{L}_i$.

- A *minimal color-spanning object* contains at least one point from each color and any sub-region of it does not contain all colors. We can assume without loss of generality that a minimal color-spanning equilateral triangle with the base parallel to x-axis (minimal CST, for short) is defined with two or three points with unique colors on its edges, as illustrated in Figure 3.

2.2 Algorithm

We now present our algorithm to compute the SCST. The sketch of our algorithm is as follows. We sweep the points with a horizontal line l from top to bottom and we test all minimal CSTs when an insertion occurs.

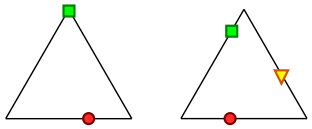


Figure 3: Two types of a minimal color-spanning equilateral triangle.

We first show a new view of a color-spanning triangle whose base lies on some horizontal line l. Let $\mathcal{P}_l \subseteq \mathcal{P}$ be the subset of points above l and $\mathcal{L} = \{\mathcal{L}_1, \cdots, \mathcal{L}_k\}$ be the set of lower envelopes of all colors for the points in $\mathcal{P}_l$. Moreover, let R be the region above the upper envelope $\mathcal{U}_\mathcal{L}$. We present the following lemma.

Lemma 1 *An equilateral triangle T whose base lies on line l is color-spanning if and only if its apex belongs to R.*

Proof. Let p' be some point in $\mathcal{P}_l$. Clearly, triangle T with base on l contains p' if and only if its apex belongs to $\mathcal{W}(p')$. Since the region above $\mathcal{L}_i$ is the union of all wedges of points with color i, the triangle T with apex located at a point above $\mathcal{L}_i$ contains at least one i-colored point. From the fact that R is the intersection of regions above $\mathcal{L}_i$ for all i, we conclude that T is color-spanning if its apex belongs to R. The converse implication can be proved in a similar way. □

Now, suppose we sweep the points with line l. When the sweep line l crosses a new point $p \in \mathcal{P}$ with color i, the insertion of p changes the structure $\mathcal{L}_i$—see Figure 4. Thus, the upper envelope may also be changed. To see the relation between minimal color-spanning triangles and the upper envelope of described structures, we present the following lemma.

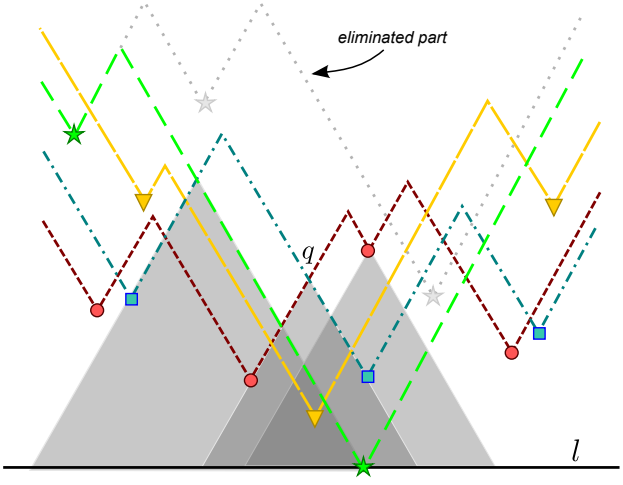


Figure 4: The apex of a minimal CST is placed on a valley point of $\mathcal{U}_\mathcal{L}$.

Lemma 2 *A triangle T with apex q is a minimal CST if and only if q appears for the first time as a valley point on some $\mathcal{U}_\mathcal{L}$ during the plane sweep.*

Proof. Consider the event that l encounters the point p with color i and q appears as a valley point on $\mathcal{U}_\mathcal{L}$ for the first time. Let T be the triangle defined by the apex q with base on l. As q belongs to $\mathcal{U}_\mathcal{L}$, triangle T is color-spanning according to Lemma 1. To conclude that T is minimal, we first show that p is the unique point with color i included in T. Since q appeared for the first time, therefore q belongs to $\mathcal{W}(p)$ and T contains p on its bottom edge. Besides, since q appeared for the first time, it must be a point below the structure $\mathcal{L}_i$ before the insertion of point p. Thus, T could not contain any point of color i just before the time l reaches p. On the other hand, each valley point of $\mathcal{U}_\mathcal{L}$ can be either a valley of some $\mathcal{L}_j$ or an intersection point of $\mathcal{L}_j$ and $\mathcal{L}_t$ for some colors j and t—see Figure 4. These cases determine the two types of minimal CSTs which are illustrated in Figure 3. We can simply prove the converse implication in a similar way, but we omit the details due to the space constraint. □

Therefore, we can state that the number of minimal CSTs is equal to the number of distinct valleys that are created on $\mathcal{U}_\mathcal{L}$ during the plane sweep process. Although there is a configuration of points $\mathcal{P}$ in which the lower envelops $\mathcal{L}_i$ intersect each other in $\Omega(nk)$ points, we show that only $O(n)$ distinct valley points can appear on $\mathcal{U}_\mathcal{L}$. Let app_i be the number of mountains that appear for the first time on $\mathcal{U}_\mathcal{L}$ when l reaches point p_i. Similarly, let del_i be the number of deleted mountains. We first present the following lemma.

Lemma 3 *The number of newly appeared valleys is at most $del_i + app_i$ when l crosses point p_i.*

Proof. We can partition the set of newly appeared mountains into m components of consecutive mountains of $\mathcal{U}_\mathcal{L}$, as illustrated in Figure 5. Each component should be placed under at least one deleted mountain. It is easy to see that the number of newly appeared valleys that are located among these components is at most $app_i + m$—see Figure 5. In addition, there could be some deleted mountains such that there is not any newly appeared mountain below them. The number of this type of deleted mountains is at most $del_i - m$. We can simply show that the number of newly appeared valleys of this type is at most $del_i - m$. Therefore, since a deleted mountain never reappears we conclude the lemma.

□

From the fact that each point p_i creates exactly two new mountains on $\mathcal{L}_i$ the total number of created mountains during the sweep process is $2n$. Moreover, each mountain can appear on $\mathcal{U}_\mathcal{L}$ and is deleted once. Thus, we conclude the following lemma according to Lomma 3.

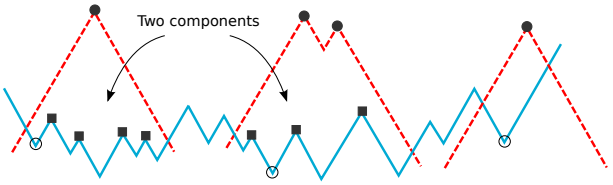


Figure 5: Squares and filled circles indicate respectively the newly appeared and the deleted mountains.

Lemma 4 *There are $O(n)$ minimal CSTs in the worst case.*

Brodal et al. [2] presented a data structure to maintain the planar maximal points with $O(\log n)$ worst case time per insertion or deletion. They define that a point $p = (p_x, p_y)$ dominates another point $q = (q_x, q_y)$ if inequalities $p_x \geq q_x$ and $p_y \geq q_y$ hold. Moreover, they showed their data structure allows reporting the maximal points that dominate a given query point q in $O(\log n + r)$ time, where r is the number of reported points.

It is easy to see that we can adopt the definition of maximal points used in this paper with the one defined in [2]. Furthermore, to maintain the upper envelope $\mathcal{U}_\mathcal{L}$ it suffices to maintain the maximal points of mountains instead of the original points. A valley on $\mathcal{U}_\mathcal{L}$ can be obtained by two consecutive maximal mountain points. We exploit the data structure presented in [2] to perform the insertion and deletion of mountains. Moreover, we use a maximal reporting query to obtain the newly appeared valleys. In the following we explain the details.

Consider the event that l passes point p with color i. First, we intersect the boundary of $\mathcal{W}(p)$ with $\mathcal{L}_i$. Let s and t be the intersected points such that $s_x \leq t_x$ (s or t may be equal to $-\infty$ and $+\infty$ respectively). Moreover, let $Q = \{q_1, \cdots, q_m\}$ be the sorted list of points that appear on $\mathcal{L}_i$ from s to t. First, we do the insertions of mountains s and t. Now, to perform the deletion of point q_i and report the newly appeared valleys we do as follows. As the first case, suppose q_i is not a point of $\mathcal{U}_\mathcal{L}$. In this case, we only perform a deletion of q_i and no new valley gets reported. In the other case, q_i is a maximal mountain point. To report the new valleys that appear by deletion of q_i, we first compute the query point q_i' by finding the two adjacent mountains of q_i on $\mathcal{U}_\mathcal{L}$—see Figure 6. Then, we perform a deletion of q_i from the data structure and perform a maximal reporting with query point q_i' to compute the newly appeared valleys. Therefore, we present the following theorem.

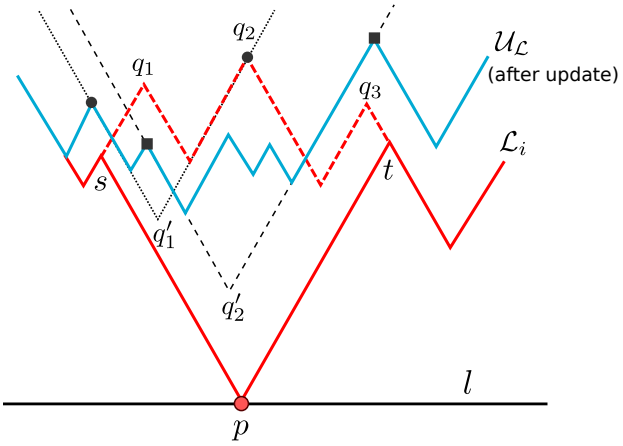


Figure 6: Updating the upper envelope $\mathcal{U}_{\mathcal{L}}$ and reporting the newly appeared valleys.

Theorem 5 *For a given set of n points with k colors in the plane, the problem of computing the SCST can be solved in $O(n \log n)$ time.*

Proof. We can compute s and t by performing a binary search in the sorted list of $\mathcal{L}_i$ in $O(\log n)$ time. Since there are totally $2n$ mountains, all insertions and deletions take $O(n \log n)$ time. On the other hand, the data structure presented in [2] allows us to compute each query point q_i' in $O(\log n)$ time. Since we perform $O(n)$ queries in such a way that each minimal CST gets reported only once, we conclude from Lemma 4 that the query takes $O(n \log n)$ time in total. Therefore, the running time of the algorithm is $O(n \log n)$. □

3 Approximating the Smallest Perimeter Color-Spanning Convex Hull

In this section, we show that the SCST gives a constant factor approximation for the smallest perimeter color-spanning convex hull (SCSCH, for short). The problem of computing the SCSCH is to choose one point from each color in such a way the perimeter of convex hull of the selected points gets minimum. We present the following theorem.

Theorem 6 *There is a 2-approximation algorithm for computing the SCSCH running in $O(n \log n)$ time.*

Proof. Let T be the SCST computed by the algorithm described in previous section. Moreover, let CH^* be the SCSCH. Since T is color-spanning there is a convex hull CH inscribed in T. Therefore, inequalities $P(CH^*) \leq P(CH) \leq P(T)$ hold, where $P(.)$ denotes the perimeter function. On the other hand, let T' be the smallest equilateral triangle which contains CH^*. For the points included in T', we can show by primitive calculations that the perimeter of any convex hull that touches the edges of T' is at least half of the perimeter of T'—consider the equilateral triangle with vertices placed at the middle points of T' edges. Therefore, inequalities $P(CH^*) \geq \frac{1}{2}P(T') \geq \frac{1}{2}P(T)$ also hold and we conclude that the SCST computes a 2-approximation of SCSCH. □

4 Conclusion

In this paper, we presented a novel idea to solve the problem of computing the SCST. We first proved that there are $O(n)$ minimal CSTs. Then, we proposed an $O(n \log n)$ time algorithm to compute the SCST which exploits the data structure presented in [2]. Finally, we showed that a 2-approximation of the SCSCH can be computed in $O(n \log n)$ time by finding the SCST which significantly improves one of the earlier results.

References

[1] Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: Smallest Color-Spanning Objects. Proc. European Symp. Algorithms (ESA-2001), LNCS - 2161, (2001) 278–289.

[2] Brodal, G. S., Tsakalidis, K.: Dynamic Planar Range Maxima Queries. In Automata, Languages and Programming. Springer Berlin Heidelberg (2011) 256–267.

[3] Gupta, P., Janardan, R., Smid, M. H. M.: Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization. J. Algorithms. **19 2** (1995) 282–317.

[4] Huttenlocher, D. P., Kedem, K., Sharir, M.: The Upper Envelope of Voronoi Surfaces and Its Applications. Discrete Computational Geometry. **9** (1993) 267–291.

[5] Ju, W., Fan, C., Luo, J., Zhu, B., Daescu, O.: On Some Geometric Problems of Color-Spanning Sets. Journal of Combinatorial Optimization. **26 2** (2013) 266–283.

[6] Khanteimouri, P., Mohades, A., Abam, M. A., Kazemi, M. R.: Computing the Smallest Color-Spanning Axis-Parallel Square. In Algorithms and Computation (ISAAC 2013), LNCS - 8283, (2013) 634–643.

[7] Löffler, M.: Data Imprecision in Computational Geometry. Ph.D. Thesis. Utrecht University. (2009).

[8] Matoušek, J.: On Enclosing k Points by a Circle. Inf. Process. Lett.. **53 4** (1995) 217–221.

Elastic Shape Matching for Translations under the Manhattan Norm

Christian Knauer* Luise Sommer* Fabian Stehn*

Abstract

The Elastic shape matching (ESM) framework is a generalization of the well-studied geometric shape matching problems. For a geometric shape matching problem, one seeks a single transformation (drawn from an appropriate class of transformations) that if applied to a geometric object (the pattern) minimizes the distance of the transformed object to another geometric object (the model).

In an ESM problem, the pattern is partitioned into parts which are transformed by a *transformation ensemble* (a collection of transformations) to minimize the distance of the individually transformed parts to the model under the constraint, that some transformations of the ensemble have to be *similar*.

We present algorithms to solve the decision variant of a ESM problem under translations for point sets under Hausdorff distance (with respect to the Manhattan metric and other polygonal metrics), if the dependencies of the transformations that are forced to be similar form a tree.

1 Introduction

The following problem and definitions will be stated for points and vectors $a \in \mathbb{R}^2$, written as $a = (a.x, a.y)$.

Definition 1 *The* directed Hausdorff distance *of a point set $A \subseteq \mathbb{R}^2$ to a point set $B \subseteq \mathbb{R}^2$ under Manhattan norm is defined as*

$$\overrightarrow{h}(A,B) := \max_{a\in A} \min_{b\in B} ||a-b||_1,$$

where $||\cdot||_1$ denotes the Manhattan norm (L_1 norm).

Problem 1 (ESM for Points under Translations)

Given:	$P = \{p_1, \ldots, p_n\}$	*point set (the pattern)*
	$Q = \{q_1, \ldots, q_m\}$	*point set (the model)*
	$G = (V, E)$	*graph with*
		$V = \{i \mid 1 \le i \le n\}$ *and*
		$E \subseteq \{\{i,j\} \mid i,j \in V\}$
	δ	*a parameter*

Question: Are there n translations $T = (t_1, \ldots, t_n)$ so that

$$\max\left(\overrightarrow{h}(T(P), Q), \max_{\{i,j\}\in E} ||t_i - t_j||_1\right) \le \delta, \qquad (1)$$

where $T(P) := \{(p_i.x + t_i.x, p_i.y + t_i.y) \mid 1 \le i \le n\}$?

The objective (Equation 1) is twofold: the task is to minimize the Hausdorff distance of the transformed pattern to the model while simultaneously minimizing the dissimilarity of translations that are paired in G. The similarity of two translations is measured by the norm of their translation vector difference.

The graph G is called *neighborhood graph* as it encodes the pairs of subpatterns that have to be matched by similar transformations. The computational complexity of a strategy to decide Problem 1 depends in the structure of G. It has been shown in [1] that the decision problem is NP-complete for translations of point sets under Hausdorff distance (with respect to the Euclidean metric), if all transformations have to be pairwise similar (G is complete). In this abstract we restrict our attention to neighborhood graphs that are trees.

ESM finds application in fields where provable precise alignments have to be computed in the presence of non-rigid deformation, such as for soft-tissue registrations for computer-guided medical interventions.

2 An Algorithm solving Problem 1

The set $I_{p,q}$ of admissible translations that move a point $p \in P$ at least δ-close to a point $q \in Q$ with respect to the Hausdorff distance under Manhattan norm can be described by a square with edge length $a := \sqrt{2}\delta$ centered in $q - p$:

$$I_{p,q} := \{t \in T \mid ||p + t - q||_1 \le \delta\}.$$

Similarly, the set $I_{p,Q}$ of translations that move a point $p \in P$ at least δ-close to some point of Q is given as $I_{p,Q} := \bigcup_{q\in Q} I_{p,q}$: a union of m squares of the same size.
We will denote a square with edge length a centered in the origin by $\diamond$.
Since $G = (V, E)$ is a tree, the algorithm starts with picking an arbitrary node $r \in V$ and henceforth considers G_r, the tree rooted in r. For internal nodes $v \in V$ let $c(v)_1, \ldots, c(v)_{n_v}$ be the children of v. For any node $v \in V$ let T_v be the subtree of G_r with root

*Institut für Angewandte Informatik, Universität Bayreuth
[christian.knauer|luise.sommer|fabian.stehn]@uni-bayreuth.de

v.
The strategy for deciding whether there is a set T of translations that satisfies Equation (1) has iterative structure. The basic idea is to propagate admissible translations from *bottom-to-top* by contracting inner nodes with their children and by appropriately merging their admissible transformations. That is, starting with G_r, the algorithm chooses an inner node and contracts it which leads to a new tree. If an internal node v has been contracted to a single node with an non-empty region $I_{v',Q}$ of admissible translations, it is possible to choose valid translations for all nodes of T_v.
In each iteration of the algorithm, we call the tree from which a node is selected the *current tree*. In each step of the algorithm, a vertex v of the current tree is selected with the property that all children of v are leafs. Then, v and the children of v are contracted to a new inner node v' which itself becomes a leaf in the resulting tree. To compute the set $I_{v',Q}$ of admissible regions for the new leaf v' we proceed as follows: first, we *inflate* all regions $I_{c(v)_i,Q}$ by δ for $1 \leq i \leq n_v$ which results in a set $I^{\delta}_{c(v)_i,Q} := I_{c(v)_i,Q} \oplus \diamond$, where $\oplus$ denotes the Minkowski sum. Note, that inflating some shifted $\diamond$ by δ leads to a square with edge length $2a$ and circumcircle with radius 2δ. The admissible regions for the new node v' are given by:

$$I_{v',Q} := \left(\bigcap_{i=1}^{n_v} I^{\delta}_{c(v)_i,Q} \right) \cap I_{v,Q}.$$

This process is iterated until either of the following two cases occurs:

1. For some node v we have $I_{v,Q} = \varnothing$:
   In this case, the process stops and *no* is returned as the answer to Problem 1.
2. The root r is contracted and $I_{r',Q} \neq \varnothing$:
   The algorithm terminates and returns *yes* as the answer to Problem 1.

The runtime of this strategy depends on the description complexity of the set of transformations that are stored in each node during the contraction of G to a single node. To analyze these sets, we need to introduce some notation.

Definition 2 *Let $B \subset \mathbb{R}^2$ be a closed connected set. A closed set $A \subset \mathbb{R}^2$ is called B-fat, if the following holds: For every point $a \in A$ there is a translation $t \in \mathbb{R}^2$ and a point $b \in B$ with $a = t + b$ and $\tilde{b} + t \in A$ for every $\tilde{b} \in B$.*

Lemma 1 ([3]) *The union of m regular k-gons has a description complexity of $O(km)$.*

Lemma 2 *Let at some point of the algorithm $c(v)_i$ with $i \in \{1, \ldots, n_v\}$ be the children of a node v which all have been contracted with their children before and hence are leafs in the current tree T_v. If every $I_{c(v)_i,Q}$ consists of m_i vertices and edges for $i \in \{1, \ldots, n_v\}$, the following holds:*

1. *Set $I_{v',Q} := \left(\bigcap_{i=1}^{n_v} I^{\delta}_{c(v)_i,Q} \right) \cap I_{v,Q}$ has description complexity $O(n_v(n_v m + \sum_{i=1}^{n_v} m_i))$.*
2. *Set $I^{\delta}_{v',Q}$ has description complexity $O(n_v m + \sum_{i=1}^{n_v} m_i)$.*

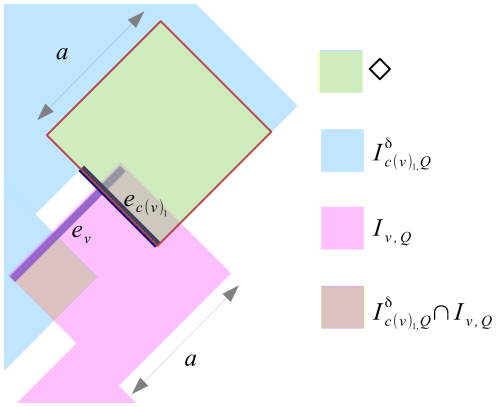


Figure 1: sets $I^{\delta}_{c(v)_1,Q}$ (blue) and $I_{v,Q}$ with focus on edges e_v and $e_{c(v)_1}$

Proof. Let all vertices have pairwise different x- and y-coordinates for now. Lets first take a closer look at the regions that constitute $I_{v',Q}$:
a) The set $I_{v,Q}$ consists of a union of shifted versions of $\diamond$ with at most $4m$ vertices and edges and covers an area of measure at most $a^2 m$; *b)* Every $I_{c(v)_i,Q}$ can be described by unions and intersections of shifted and inflated versions of $\diamond$ and has a surface with size up to $a^2 m$ since it has either already been a leaf of T_v in the beginning or is the result of a contraction of other nodes. Hence every $I^{\delta}_{c(v)_i,Q}$ covers an area of measure up to $4a^2 m$.
Each set can be described by a list of its boundary edges where every edge is of one of four types: it can have a slope of ± 1 and the interior of the set can either be above or below the edge.
Every edge in $I_{v',Q}$ either originates from an edge of $I_{v,Q}$ or from an edge of $I^{\delta}_{c(v)_i,Q}$ for some i. Since the description complexity of $I_{v',Q}$ is equal to the maximum number of its boundary edges, we can count in how many parts the edges of $I^{\delta}_{c(v)_i,Q}$ and $I_{v,Q}$ can be cut in order to determine its complexity.
The sets $I_{v,Q}$ and all $I^{\delta}_{c(v)_i,Q}$ are obviously $\diamond$-fat and as $I_{v,Q}$ is a collection of m pseudo discs and hence has a linear description complexity, see Lemma 1.

Part 1 of Lemma 2: Consider an arbitrary edge e_v of $I_{v,Q}$: e_v can only be intersected by edges of $I^{\delta}_{c(v)_i,Q}$ that are orthogonal to e_v. Let $e_{c(v)_i}$ be such an edge intersecting e_v in s. Since $I^{\delta}_{c(v)_i,Q}$ is $\diamond$-fat

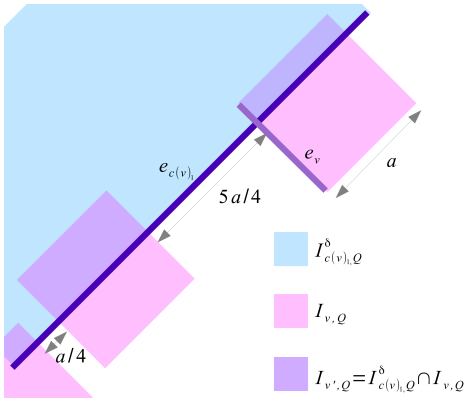


Figure 2: intersecting $I^{\delta}_{c(v)_1,Q}$ and $I_{v,Q}$ with focus on edge $e_{c(v)_1}$

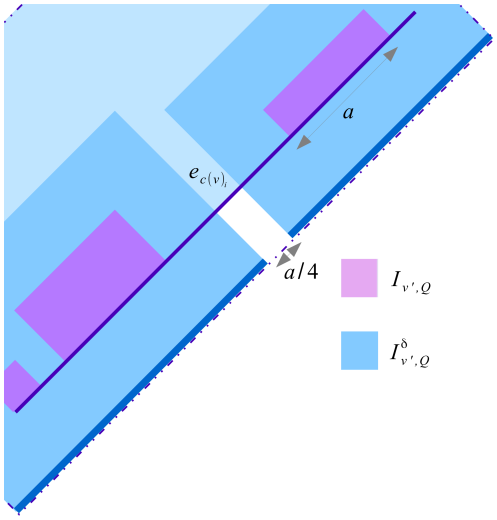


Figure 3: inflating $I_{v',Q}$ with focus on the magenta and resulting blue edges

we can place a square $\diamond_*$ of side length a so that it has s on its boundary and is completely contained in $I^{\delta}_{c(v)_i,Q}$. As e_v has a length of a and $\diamond_*$ has a side length of a, one endpoint of e_v has to be in $\diamond_*$, see Figure 1. As a consequence, e_v can be cut in at most two pieces by edges of $I^{\delta}_{c(v)_i,Q}$. Since $\left(\bigcap_{i=1}^{n_v} I^{\delta}_{c(v)_i,Q}\right) \cap e_v = \bigcap_{i=1}^{n_v}(e_v \cap I^{\delta}_{c(v)_i,Q})$ and the parts of e_v that remain after intersection with $I^{\delta}_{c(v)_i,Q}$ resemble intervals, the maximum number of pieces of $\left(\bigcap_{i=1}^{n_v} I^{\delta}_{c(v)_i,Q}\right)\cap e_v$ is the sum of the number of pieces of all $e_v \cap I^{\delta}_{c(v)_i,Q}$, which is $2n_v$. Hence there are at most $8n_vm$ edges in $I_{v',Q}$ that originate from an edge of $I_{v,Q}$.
By similar arguments, an edge of $I^{\delta}_{c(v)_i,Q}$ of length k can be cut by $I_{v,Q}$ in at most $1+\lceil\frac{k}{a}\rceil$ disjoined pieces that remain as edges in $I_{v',Q}$, see Figure 2. As every $I^{\delta}_{c(v)_j,Q}$ with $i \neq j$ is $\diamond$-fat as well, the same holds for intersecting $I^{\delta}_{c(v)_i,Q}$ with some $I^{\delta}_{c(v)_j,Q}$. As $I^{\delta}_{c(v)_i,Q}$ is $\diamond$-fat and the size of its surface is at most $4a^2m$ the sum of the lengths of all edges of $I^{\delta}_{c(v)_i,Q}$ which are of the same type is less than $\frac{4a^2m}{a} = 4am$. To get the maximum amount of edges in $I_{v',Q}$ that originate from edges of $I^{\delta}_{c(v)_i,Q}$ we assume that $I^{\delta}_{c(v)_i,Q}$ has $m_i - 4$ very short edges which cause at most two edges in $I_{v',Q}$ and four long edges with length at most $4am$. Therefore the maximum number of edges in $I_{v',Q}$ originating from edges of some $I^{\delta}_{c(v)_i,Q}$ is $n_v(2m_i + 16m - 4)$, which gives a bound on the number of edges in $I_{v',Q}$ of

$$4n_v^2(4m-1) + 2n_v(4m + \sum_{i=1}^{n_v} m_i), \qquad (2)$$

which proves part 1.

Part 2 of Lemma 2: Let $e'_{c(v)_i}$ and $e''_{c(v)_i}$ be two edges in $I_{v',Q}$ which result from cutting the same edge $e_{c(v)_i}$ of $I^{\delta}_{c(v)_i,Q}$. Inflating $I_{v',Q}$ by δ in Manhattan norm equates to adding a shifted version of $\diamond$ centered at all points of its boundary since $I^{\delta}_{v',Q} = I_{v',Q} \oplus \diamond$. Let e_v be some boundary edge of $I_{v',Q}$, its corresponding boundary edge e'_v of $I^{\delta}_{v',Q}$ will be shifted parallel by an amount of $\frac{a}{2}$ and will be extended by $\frac{a}{2}$ at both ends. Therefore $e'_{c(v)_i}$ and $e''_{c(v)_i}$ will merge in $I^{\delta}_{v',Q}$ to one edge, if they are no more than a apart in $I_{v',Q}$ (the same arguments hold for edges of $I_{v,Q}$), see Figure 3. This fact includes that every edge of $I^{\delta}_{c(v)_i,Q}$ with length k can cause at most $\lceil\frac{k}{a}\rceil$ edges in $I^{\delta}_{v',Q}$ and every edge of $I_{v,Q}$ can cause at most one edge in $I^{\delta}_{v',Q}$, no matter in how many pieces it has been cut before.
We again use the fact that $I^{\delta}_{c(v)_i,Q}$ is $\diamond$-fat and covers an area of measure at most $4a^2m$: The largest number of edges in $I^{\delta}_{v',Q}$ that originate from edges of $I^{\delta}_{c(v)_i,Q}$ is achieved when $I^{\delta}_{c(v)_i,Q}$ has $m_i - 4$ short edges each causing at most one edge in $I^{\delta}_{v',Q}$ and four long edges of length at most $4am$ each causing up to $4m$ new edges.
Summing up all these edges results in at most $m_i - 4 + 16m$ edges that arise from edges of $I^{\delta}_{c(v)_i,Q}$ for every $1 \leq i \leq n_v$ and $4m$ edges with source $I_{v,Q}$ in $I^{\delta}_{v',Q}$. Hence the maximum number of edges in $I^{\delta}_{v',Q}$ is

$$4n_v(4m-1) + 4m + \sum_{i=1}^{n_v} m_i. \qquad (3)$$

This proves part 2.

Please note that dropping the general position assumption of the vertices of $I_{c(v)_i,Q}$ and $I_{v,Q}$ has the only effect that (close) edges of the same type may merge during the process, which only shortens the largest possible length of edges in the above arguments. □

Theorem 3 *Problem 1 can be decided in*

$O(n^2m(\log m + \log n))$ time for neighborhood graphs that are trees (also when reporting a witness for a yes-instance).

Proof. The first step is to compute the regions $I_{p,Q}$ for all $p \in P$, the initial admissible translations stored in the nodes of G. These sets are represented by the boundary edges of squares centered in $q - p$ for each $q \in Q$. It takes $O(m)$ time to determine $I_{p,Q}$ for a fixed p. Since the squares form a collection of pseudodiscs their union has description complexity $O(m)$ which can be computed in a sweep line manner in $O(m \log m)$ time.
Let T_v be some subtree of T_r of k_v nodes and $v \neq r$. We let the algorithm run on this subtree until it stops or all nodes have been contracted to a new node v' and the resulting set $I_{v',Q}$ has been inflated to $I^{\delta}_{v',Q}$. This set has description complexity $O(k_v m)$ according to Lemma 2, recursive application of equation (3). Each child of the root r has a corresponding subtree $T_{c(r)_i}$ containing k_i nodes for $1 \leq i \leq n_r$. Consider one fixed $c(r)_i$. Applying the algorithm on $T_{c(r)_i}$ and inflating the admissible regions of its root results in a set $I^{\delta}_{c(r)'_i}$ of admissible translations for all nodes of $T_{c(r)_i}$. The set $I^{\delta}_{c(r)'_i}$ is $\diamond$-fat and has description complexity $O(k_i m)$ due to Lemma 2, part 2. After the last step of the algorithm, the description complexity of the set of admissible translations stored in r is $O(n_r^2 m + n_r nm) = O(n^2 m)$, as $\sum_{i=1}^{n_r} k_i = n - 1$, due to the first part of Lemma 2.

The total runtime of the algorithm stems from the time that is needed to carry out the union/intersection operations to compute the intermediate admissible transformations $I_{v',Q}$ for all nodes v of T_r. According to the first part of Lemma 2, $I_{v',Q}$ has a complexity of $O(n_v^2 m + n_v nm)$ (where n_v is the number of children of v), as each $I^{\delta}_{c(v)_i}$ has description complexity $O(nm)$.

Adding up the sizes of these sets over all nodes v of T_r for wich these admissible translations are computed during the algorithm results in a total complexity of $O(n^2 m)$. The sum of the runtime that has to be spend in each node to perform the union/intersection operations on sets of size $O(n_v^2 m + n_v nm)$ can be bounded from above by considering the respective runtime for a set of total size $O(n^2 m)$, hence we can conclude that the total runtime of the algorithm is $O(n^2 m(\log m + \log n))$. □

Note, that all results and arguments also hold when considering the L_∞ norm instead of the L_1 norm. In many ESM applications, considering the Hausdorff distance under the Euclidean norm seems to be a more natural distance measure than the Hausdorff distance under L_1 norm. Indeed, a very similar strategy (computing, inflating and propagating admissible regions from bottom-to-top) can be applied to decide Problem 1 in this setting. However, inflating regions with a L_2 ball (by computing the Minkowski sum) instead of a L_1 ball increases the complexity of the respective regions and hence makes it hard to analyze the runtime of this method.

But as a L_1 ball of radius δ is contained in a L_2 ball of that radius, we can use the above algorithm to approximate the Euclidean setting as formulated in Corollary 4.

Corollary 4 *The algorithm discussed above gives an $\sqrt{2}$-approximation for Problem 1 with same settings and under Euclidean norm.*

Approximating here means that if $\delta^{opt}_{L_1}$ is the optimal (smallest) value in the L_1 setting, the optimal value $\delta^{opt}_{L_2}$ in the Euclidean setting is bounded by

$$\delta^{opt}_{L_2} \leq \delta^{opt}_{L_1} \leq \sqrt{2}\,\delta^{opt}_{L_2}.$$

It is easy to see that the complexity of inflated admissible regions does not increase when the unit ball of the underlying metric is a regular polygon. This allows to improve upon the approximation factor: Let $\mathcal{P}_{2b}$ be a regular polygon with $2b \geq 4$ vertices, $b \in \mathbb{N}$, centered in the origin with radius 1. $\mathcal{P}_{2b}$ is centrally-symmetric as it has an even number of corners and induces a norm and hence a metric to which we refer as the $\mathcal{P}_{2b}$-metric, see [2] for further information.

Theorem 5 *Problem 1 under $\mathcal{P}_{2b}$-metric can be decided in $O(b^2 n^3 m^2(\log m + \log n + \log b))$ time (also when reporting a witness for a yes-instance).*

Due to space limitations we skip the proof of Theorem 5.

Theorem 6 *Given an integer $k \geq 2$ and $b := \left\lceil \frac{\pi}{4}\sqrt{2k} \right\rceil$ Problem 1 under $\mathcal{P}_{2b}$-metric gives an $\left(1 + \frac{1}{k}\right)$-approximation for Problem 1 with same settings and under Euclidean norm.*

Due to space limitations we skip the proof of Theorem 6.

References

[1] Knauer, Christian; Stehn, Fabian: *Elastic Shape Matching is NP-hard for Point Sets under Hausdorff Distance.* In Proc. 30th European Workshop on Computational Geometry (EuroCG'14), 2014.

[2] Barvinok, Alexander; Johnson, David S.; Woeginger, Gerhard J.; Woodroofe, Russell: *Finding Maximum Length Tours Under Polyhedral Norms.* Proceedings of IPCO VI, Lecture Notes in Computer Science 1412, 1998.

[3] Agarwal, Pankaj K.; Pach, János; Sharir, Micha: *State of the Union (of Geometric Objects): A Review.* Discrete & Computational Geometry, 2007.

Combinatorics of edge 2-transmitter art gallery problems

Sarah Cannon* Thomas G. Fai† Justin Iwerks‡ Undine Leopold§ Christiane Schmidt¶

Abstract

We give sufficiency and necessity results for edge 2-transmitters in general, monotone, orthogonal and monotone, and orthogonal polygons.

1 Introduction

The traditional art gallery problem (AGP) considers placing guards in an art gallery—modeled by a polygon—so that every point in the room can be seen by some guard. A similar question asks how to place wireless routers so that an entire room has a strong signal. Observation shows that often not only the distance from a modem, but also the number of walls a signal has to pass through, influences signal strength.

Aichholzer et al. [1] first formalized this problem by considering *k-modems* (*k-transmitters*), devices whose signal can pass through at most k walls. Analogous to the AGP, two main questions can be considered: (1) Given a polygon P, can a minimum cardinality k-transmitter cover be computed efficiently? (2) Given a class of polygons of n vertices, what are lower and upper bounds on the number of guards needed to cover a polygon from this class?

For the classical AGP, the complexity question (1) was answered with NP-hardness for many variants: Lee and Lin [8] gave the result for simple polygons. In [4], we show the minimum point $2/k$-transmitter and edge 2-transmitter problems are NP-hard. Answers to (2) are often referred to as "Art Gallery theorems", e.g., Chvátal's tight bound of $\lfloor \frac{n}{3} \rfloor$ for simple polygons [5]. For k-transmitters ($k$tr), Aichholzer et al. [1] showed $\lceil \frac{n}{2k} \rceil$ $k$tr are always sufficient and $\lceil \frac{n}{2k+4} \rceil$ $k$tr are sometimes necessary to cover a monotone n-gon[1]; for monotone orthogonal polygons, they gave a tight bound of $\lceil \frac{n-2}{2k+4} \rceil$ $k$tr. Fabila-Monroy et al. [6] improved the bounds on monotone polygons to a tight value of $\lceil \frac{n-2}{2k+3} \rceil$ (k even). Other publications explored k-transmitter coverage of regions other than simple polygons, such as coverage of the plane in the presence of line or line segment obstacles [2, 7].

For the classical AGP, variants were considered: e.g., *edge guards* that monitor each point of the polygon that is visible to some point of the edge. Bjorling-Sachs [3] showed a tight bound of $\lfloor \frac{3n+4}{16} \rfloor$ edge guards for rectlinear polygons. For general polygons $\lfloor \frac{3n}{10} \rfloor + 1$ edge guards are always sufficient and $\lfloor \frac{n}{4} \rfloor$ are sometimes necessary [10].

Polygon Class	Always Sufficient	Sometimes Necessary
General	$\lfloor \frac{3n}{10} \rfloor + 1$ [10]	$\lfloor \frac{n}{6} \rfloor$ (Th. 5)
Monotone	$\lceil \frac{(n-3)}{8} \rceil$ (Th. 9)	$\lceil \frac{(n-2)}{9} \rceil$ (Th. 6)
Mon. Orth.	$\lceil \frac{(n-2)}{10} \rceil$ (Th. 13)	$\lceil \frac{(n-2)}{10} \rceil$ (Th. 10)
Orthogonal	$\lfloor \frac{(3n+4)}{16} \rfloor$ [3]	$\lceil \frac{(n-2)}{10} \rceil$ (Th. 10)

Table 1: Results for edge 2-transmitters in simple n-gons.

Our Results are summarized in Table 1. We consider simple polygons only (of course, the necessity results transfer to the case of polygons with holes).

2 Notations and Preliminaries.

A point $q \in P$ is *2-visible* from $p \in P$ if the straight-line connection $\overline{pq}$ intersects P in at most two connected components. For a point $p \in P$, we define the *2-visibility region* of p, $2\mathrm{VR}(p)$, as the set of points in P that are 2-visible from p. For a set $S \subseteq P$, $2\mathrm{VR}(S) := \cup_{p \in S} 2\mathrm{VR}(p)$. A set $C \subseteq P$ is a *2-transmitter cover* if $2\mathrm{VR}(C) = P$. Points used for a 2-transmitter cover are called *(point) 2-transmitters*. An *edge 2-transmitter* e can monitor all points of P that are 2-visible from some $q \in e$.

3 Point 2-transmitters

We start with observations on point 2-transmitter covers that enable the edge 2-transmitter results. Some proofs are omitted due to space limits.

Lemma 1 *Every 5-gon can be covered by a point 2-transmitter placed anywhere (boundary or interior).*

Lemma 2 *Let P be a 6-gon, $e = \{v, w\}$ an edge of P. A point 2-transmitter at v or at w covers P.*

*College of Computing, Georgia Institute of Technology, Atlanta, USA. E-mail: sarah.cannon@gatech.edu. Supported by a Clare Boothe Luce Graduate Fellowship and NSF DGE-1148903.

†School of Engineering and Applied Sciences, Harvard University, MA, USA. E-mail: tfai@seas.harvard.edu

‡Mathematics Department, The Spence School, NY, USA. E-mail: jiwerks@gmail.com

§Mathematics Department, TU Chemnitz, Germany. E-mail: undine.leopold@mathematik.tu-chemnitz.de

¶The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem. E-mail: cschmidt@cs.huji.ac.il. Supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

[1] The stated lower bound of $\lceil n/(2k+2) \rceil$ given in [1] is a typo, and the example only necessitates $\lceil n/(2k+4) \rceil$ 2-transmitters.

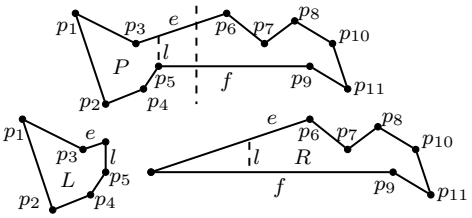


Figure 1: Application of the Splitting Lemma of [1] to an 11-gon P for $m = 6$, resulting in a 6-gon L and a 7-gon R.

Proof. Let $e = \{v, w\}$. By the Two Ear Theorem [9], there exists a diagonal from either v or w that splits off a triangle T from P. Removing T leaves a 5-gon P' which has v or w as one of its vertices; call this vertex $\tilde{v}$. The addition of T does not increase the number of connected components (CC) of the polygon on any visibility ray starting at $\tilde{v}$. That is, for any ray r starting at $\tilde{v}$, the number of CCs of $r \cap P'$ is the same as the number of CCs of $r \cap P$. Thus, all of P is visible from a 2-transmitter on $\tilde{v}$. □

Lemma 3 *Every monotone 6-gon can be covered by a single (point) 2-transmitter placed at one of its two leftmost (or rightmost) vertices.*

Lemma 4 (Splitting Lemma, [1]) *Let P be a monotone polygon with vertices p_1, p_2,..., p_n, ordered from left to right. For every positive integer $m < n$, there exists a vertical line segment l and two monotone polygons L and R such that*

- *L has m vertices and R has $n - m + 2$ vertices.*
- *Either l is a chord of L and an edge of R, or l is an edge of L and a chord of R.*
- *p_m or p_{m+1} is an endpoint of l.*
- *Denote as L' the subset of L left of l and denote as R' the subset of R right of l; then $P = L' \cup R'$.*

Proof Sketch. Consider a vertical line intersecting P between p_{m-1} and p_m. The edges e and f this vertical line crosses, when extended, meet to either the right or left of this line (assuming they are not parallel). If they meet to the left, L and R are as in Fig. 1; otherwise, the construction of L and R is reversed.

4 Edge 2-transmitters

4.1 General Polygons

For general n-gons, the upper bound of $\lfloor \frac{3n}{10} \rfloor + 1$ edge guards from [10] obviously holds for more powerful edge 2-transmitters. The polygon requiring $\lfloor \frac{n}{4} \rfloor$ edge guards only necessitates $\lfloor \frac{n}{8} \rfloor$ edge 2-transmitters, and next we improve on this lower bound:

Theorem 5 *There exist simple n-gons that require $\lfloor n/6 \rfloor$ edge 2-transmitters.*

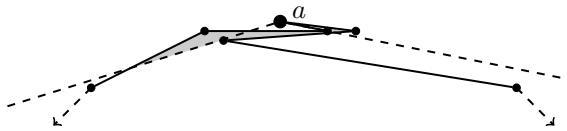


Figure 2: Lower bound construction for general polygons.

Proof. We provide a polygon P_n where any valid covering by edge 2-transmitters requires at least $\lfloor n/6 \rfloor$ edges. Fig. 2 depicts a six-edge gadget. The dashed arrows indicate the beginnings of edges of neighboring gadgets; one can arrange $\lfloor n/6 \rfloor$ of these gadgets sequentially around a circle, subdividing up to 5 edges if necessary so that P_n has n vertices and n edges. (Modified angles allow for an arbitarily large number to be placed around a circle.) Vertex a is only 2-visible from one of the six edges in the gadget (if all other gadgets remain entirely below the dashed 2-visibility lines). Thus, at least one edge from each gadget must be included in any valid edge 2-transmitter cover. □

4.2 Monotone Polygons

Theorem 6 *There exist monotone n-gons that require $\lceil (n-2)/9 \rceil$ edge 2-transmitters.*

Proof. Consider polygon P in Fig. 3. Two points $p_i, p_j \in int(P)$ within ε from a_i, a_j are not 2-visible from the same edge. So, each a_i requires an edge for coverage. $|V(P)| = 9\ell - 6$ and ℓ edge 2-transmitters are necessary. For any value of n, the construction (possibly with up to 8 edges subdivided) necessitates $\lfloor (n+6)/9 \rfloor = \lceil (n-2)/9 \rceil$ edge 2-transmitters. □

Before we can prove the upper bound for monotone n-gons, we present a crucial lemma.

Lemma 7 *Any monotone 10-gon P can be covered by a single edge 2-transmitter e, and for every point $p \in P$, there exists q on e such that p is 2-visible from q, where q is left of at least two vertices of P and right of at least two vertices of P.*

Proof. Label vertices of P left to right: $v_1, \ldots, v_{10}$. Assume no two vertices are on the same vertical line (otherwise perturb one by some sufficiently small ε).

Draw vertical line l_5 though v_5. Let e be the edge it hits on the other monotone chain; its endpoints are $a = v_i$ where $i \leq 4$ and $b = v_j$ where $j \geq 6$. First consider the vertical line l_4 through v_4. This separates from P a 5-gon $P_L \subset P$ with vertices v_1, v_2, v_3, v_4, and the other intersection of l_4 with P's boundary. By

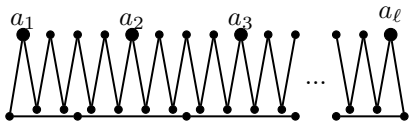


Figure 3: Lower bound for monotone polygons.

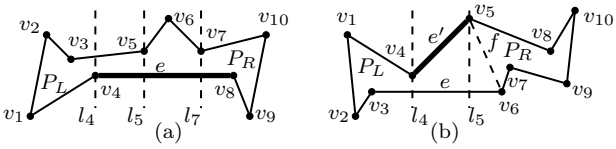


Figure 4: Examples of a monotone 10-gon P; edges covering P are thickened. (a) case 1, (b) case 2, proof of Lemma 7.

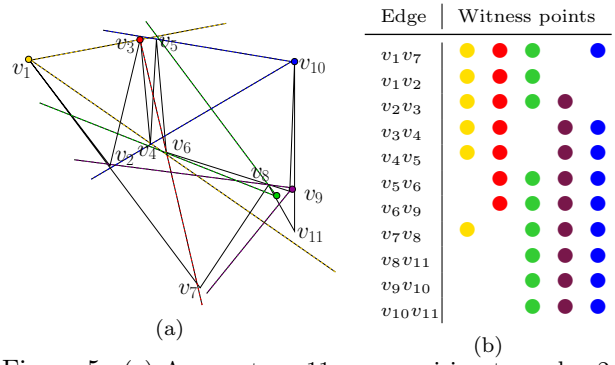


Figure 5: (a) A monotone 11-gon requiring two edge 2-transmitters; boundaries of 2-VRs of witness points are dotted. (b) • denotes the edge sees the gold witness point, etc. No edge 2-transmitter covers all five witness points.

Lemma 1, P_L can be covered by a point 2-transmitter placed anywhere in P_L, and in particular anywhere along the intersection of l_4 with P.

Case 1: If $b = v_j$ and $j \geq 7$, the vertical line l_7 through v_7 cuts a 5-gon P_R from P, with vertices v_7, v_8, v_9, v_{10}, and the other intersection of l_7 with P's boundary; see Fig. 4(a). By Lemma 1, P_R can be covered by a 2-transmitter placed anywhere in P_R. Then P_L is 2-visible from $e \cap l_4$, P_R is 2-visible from $e \cap l_7$ and the interior of e covers $P \setminus (P_L \cup P_R)$.

Case 2: Otherwise, $b = v_6$; see Fig. 4(b). Consider the line segment f connecting v_5 and v_6. We have $f \subseteq P$ and it crosses from one monotone chain to the other. To its right f seperates a 6-gon from P: P_R. By Lemma 3, P_R is 2-visible from a point 2-transmitter placed at c, $c = v_5$ or $c = v_6$. P's edge e' with right endpoint c (possibly $e' = e$) has left endpoint in the set $\{v_1, v_2, v_3, v_4\}$. Thus, e' covers P: P_R is 2-visible from c, P_L is 2-visible from $e' \cap l_4$, and the interior of e' covers $P \setminus (P_L \cup P_R)$. □

The bound of 10 is the best we can hope for: the 11-gon from Fig. 5 necessitates two edge 2-transmitters.

The next lemma follows immediately from the proof of the Splitting Lemma in [1], and is crucial to the subsequent sufficiency result.

Lemma 8 *Let P, L, R, L', R', and l be as in the Splitting Lemma. Then for every edge $e \neq l$ of L, the subset of e left of l is a subset of an edge of P. For every edge $e \neq l$ of R, the subset of e right of l is a subset of an edge of P.*

Figure 6: Lower bound for monotone orthogonal polygons. The 2-visibility region of the bold edge is shaded.

Theorem 9 *$\lceil (n-2)/8 \rceil$ edge 2-transmitters are always sufficient to cover a monotone n-gon with $n \geq 4$.*

Proof. We induct on n. For the base case, one edge 2-transmitter e covers monotone k-gons, $k = 3, 4, ..., 10$ by Lemma's 1, 3, and 7. Each $p \in P$ is 2-visible from some $q \in e$, with q to the right of at least two vertices of P.

Suppose $n > 10$ and for all $n' < n$, every monotone n'-gon P' can be covered by a set C of $\lceil \frac{n'-2}{8} \rceil$ edge 2-transmitters, and each $p \in P'$ is 2-visbile from some point $q \in e \in C$, with q right of at least two vertices of P'. Apply the Splitting Lemma for $m = 10$ to obtain a monotone 10-gon L and a monotone $(n-8)$-gon R. Let l, L', and R' be as in the lemma. Then by Lemma 7, 10-gon L can be covered by a single edge 2-transmitter e such that every point $p \in L$ is 2-visible from some point q on e that is left of at least two vertices of L and right of at least two vertices of L. Thus, $e \neq l$, and the portion of e consisting of all such points q is entirely left of l, so by Lemma 8 is a subset of an edge of P. Thus, there exists a single edge 2-transmitter t of P that covers L'. Moreover, all points in L' are 2-visible from a point $q \in t \in P$, where q is right of at least two vertices of P, because the same statement holds for edge e of L.

By the induction hypothesis, monotone $(n-8)$-gon R can be covered by a collection C of $\lceil \frac{(n-8)-2}{8} \rceil$ edge 2-transmitters, and every point in R' is 2-visible from some point q on an edge of C, where q is right of at least two vertices of R. Hence, for each edge $e \in C$, the portion t of e consisting of all such points is entirely right of l, so by Lemma 8, t is a subset of an edge of P. Thus, there exists a collection C' of $\lceil \frac{(n-8)-2}{8} \rceil$ edge 2-transmitters covering R'.

So, P has an edge 2-transmitter cover C of size $1 + \lceil \frac{(n-8)-2}{8} \rceil = \lceil \frac{n-2}{8} \rceil$ with the assumed property. □

4.3 Monotone Orthogonal Polygons

We note the following without proof; see Fig. 6.

Theorem 10 *There exist monotone orthogonal (MO) n-gons that require $\lceil \frac{n-2}{10} \rceil$ edge 2-transmitters.*

We now proceed to show the upper bound.

Lemma 11 *Any monotone orthogonal 6-gon is covered by a (point) 2-transmitter placed anywhere.*

Lemma 12 *Any monotone orthogonal 12-gon P can be covered by one edge 2-transmitter, not placed on its leftmost or rightmost edge.*

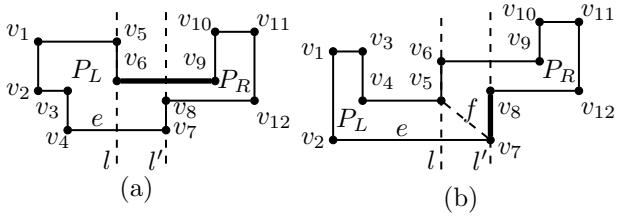


Figure 7: Examples of a MO 12-gon P; edges covering P are thickened. (a) case 1, (b) case 2, proof of Lemma 12.

Proof. Assume no two vertical edges in P have the same x-coordinate. Order the vertices of P left to right, order vertices with the same x-value (endpoints of a vertical edge e_v) according to a left to right traversal of the monotone chain containing e_v. The two leftmost and rightmost vertices can be put in any order. Label the vertices as v_1, v_2,..., v_{12} in this order, see Fig. 7. Vertical edges have endpoints v_{2i-1}, v_{2_i} for $i = 1, 2, ..., 6$. For horizontal edges (except for rightmost and left most vertices), the right (left) vertex has odd (even) index, due to monotonicity.

Consider the supporting line l through the vertical edge $\{v_5, v_6\}$. This separates from P a 6-gon with vertices v_1, v_2, v_3, v_4, v_5, and $l \cap e$, where e is an edge in the opposite monotone chain from v_5. We have $v_6 \subset l$, though it may lie on the boundary of P_L, as in Fig. 7(a), or not, as in Fig. 7(b). The supporting line l' through the vertical edge with $\{v_7, v_8\}$ also separates from P a 6-gon P_R with vertices v_8, v_9, v_{10}, v_{11}, v_{12} and $l' \cap e'$, where e' is an edge in the opposite chain from v_8. Based on v_6's location, there are two cases for the number of edges extending rightward beyond l whose left endpoint is in P_L. Note edge e always extends rightward past the right boundary of P_L, implying its right endpoint is v_i for $i \geq 7$.

Case 1: If v_6 is on the boundary of P_L, a second edge extends rightward from v_6, also with its left endpoint in P_L and its right endpoint some v_i for $i \geq 7$; see Fig. 7(a). In this case at least one of these two edges has right endpoint v_i for $i \geq 8$, so it covers P_R, P_L and the rectangular region $P \setminus (P_L \cup P_R)$.

Case 2: Else, v_6 is not on the boundary of P_L, and e is the only edge extending rightward from P_L; see Fig. 7(b). Similar to case 1, if e has right endpoint v_i for $i \geq 8$ we are done, so, we suppose v_7 is e's right endpoint. In this case, e might not cover the entirety of P_R. Let f be the segment connecting v_5 and v_7. We have $f \subset P$. We note that f separates from P a (non-orthogonal) 6-gon P'_L with vertices $v_1, v_2, v_3, v_4, v_5, v_7$, and $P_L \subseteq P'_L$. P'_L contains at most one reflex vertex and, thus, can be covered by a point 2-transmitter placed anywhere on its boundary. In particular, the edge v_7v_8 covers P'_L from v_7, P_R from v_8, and $P \setminus (P_L \cup P_R)$ from its interior.

In neither case above do we pick the rightmost or leftmost edge to cover P. □

Theorem 13 *$\lceil \frac{n-2}{10} \rceil$ edge 2-transmitters are always sufficient to cover a monotone orthogonal n-gon.*

Proof. We induct on n. For a MO n-gon P, n is even and P has $n/2$ vertical and horizontal edges. By Lemma 12, all MO m-gons with $4 \leq m \leq 12$ can be covered by one edge 2-transmitter, not placed on its leftmost edge.

Label the vertical edges of P in order left to right as $e_1, e_2, \ldots, e_{n/2}$. Consider the supporting line of edge e_6; this separates from P a MO 12-gon Q with six vertical edges $e_1, e_2, \ldots, e_5$, and some segment of the supporting line of e_6. Polygon Q can be covered by a single edge 2-transmitter, not placed on its leftmost or rightmost edge. The remainder $P \setminus Q$ has $n/2 - 5 \geq 2$ vertical edges $e_7, e_8, \ldots, e_{n/2}$, and some segment of the supporting line of e_6, so, $|V(P \setminus Q)| = n - 10$. By the inductive hypothesis, it can be covered by $\lceil \frac{(n-10)-2}{10} \rceil$ edge 2-transmitters, none of which are placed on its leftmost edge. Together with the single edge covering Q, this yields a cover of P by $1 + \lceil \frac{(n-10)-2}{10} \rceil = \lceil \frac{n-2}{10} \rceil$ edge 2-transmitters, not including P's leftmost edge. □

Acknowledgments. We would like to thank Joseph O'Rourke for sharing the 2-transmitter problem with us during the Mathematics Research Communities workshop on Discrete and Computational Geometry in 2012 and the AMS for their financial support of this program.

References

[1] O. Aichholzer, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, J. Urrutia, and B. Vogtenhuber. Modem illumination of monotone polygons. In *Proc. 25th Europ. Workshop Comp. Geom.*, pages 167–170, 2009.

[2] B. Ballinger, N. Benbernou, P. Bose, M. Damian, E. Demaine, V. Dujmović, R. Flatland, F. Hurtado, J. Iacono, A. Lubiw, P. Morin, V. Sacristán, D. Souvaine, and R. Uehara. Coverage with k-transmitters in the presence of obstacles. In W. Wu and O. Daescu, editors, *Comb. Opt. and Appl.*, volume 6509 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2010.

[3] I. Bjorling-Sachs. Edge guards in rectilinear polygons. *Comput. Geom.*, 11(2):111–123, 1998.

[4] S. Cannon, T. Fai, J. Iwerks, U. Leopold, and C. Schmidt. NP-hardness proofs for point and edge 2-transmitters. In *24th Fall Workshop on Computational Geometry*, 2014.

[5] V. Chvátal. A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B*, 18:39–41, 1975.

[6] R. Fabila-Monroy, D. Flores-Peñaloza, J. Urrutia, and B. Vogtenhuber. Personal communication, 2014.

[7] R. Fabila-Monroy, A. Vargas, and J. Urrutia. On modem illumination problems. In *XIII Encuentros de Geometria Computacional, Zaragoza*. 2009.

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

[9] G. H. Meisters. Polygons have ears. *Amer. Math. Monthly*, 82:648–651, 1975.

[10] T. C. Shermer. Recent results in art galleries. In *Proceedings of the IEEE*, volume 80, pages 1384–1399, 1992.

Chromatic Guarding of Orthogonal Polygons with Orthogonal Visibility

Frank Hoffmann* Klaus Kriegel* Subhash Suri† Kevin Verbeek‡ Max Willert*

Abstract

We address both the strong (introduced in [3]) and the conflict-free ([1]) chromatic version of the classic Art Gallery Problem. Assume a simple orthogonal polygon P is guarded by a finite set of point guards and each guard is assigned one of t colors. Such a chromatic guarding is said to be conflict-free if each point $p \in P$ sees at least one guard with a unique color among all guards visible from p. The guarding is strong if all guards visible from a point have different colors. The goal is to establish bounds on the numbers $\chi_{cf}(n)$ and $\chi_{st}(n)$ of colors sufficient to guarantee the existence of a conflict-free, respectively strong chromatic guarding for any n-vertex polygon. In this paper, we assume the r-visibility model instead of standard line visibility. Points p and q in an orthogonal polygon are r-visible to each other if the axis-parallel rectangle spanned by the points is contained in P. For this model we show tight bounds on the number of colors: $\Theta(\log\log n)$ for conflict-free and $\Theta(\log n)$ for strong guarding. Our results can be interpreted as coloring results for special geometric hypergraphs [5].

1 Preliminaries

We study simple orthogonal polygons, i.e., polygons consisting of alternating vertical and horizontal edges only. By $|P|$ we denote the number of vertices, by ∂P the boundary and by $\text{int}P = P \setminus \partial P$ the interior of the polygon. Vertices can be reflex or convex. A *reflex* vertex has an interior angle $3\pi/2$ while *convex* vertices have an interior angle of $\pi/2$. We do not make any general position assumption for the simple orthogonal polygons P. Points $p, q \in P$ are *r-visible* to each other if the closed axis-parallel rectangle $r[p, q]$ with diagonal pq is contained in P. In the following, visible always means r-visible. $V(p) = \{q \in P | r[p, q] \subseteq P\}$, the set of all points visible from p, is the *visibility polygon* of p. A polygon that is fully visible from one of its points is called a *star*. For $P' \subset P$ we define its visibility polygon by $V(P') = \cup_{p \in P'} V(p)$. The *windows* of a subpolygon P' in P are those parts of $\partial P'$ that do not belong to ∂P.

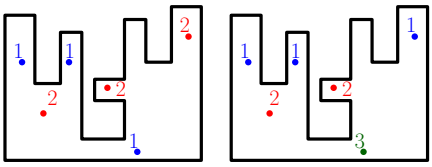


Figure 1: Example of conflict-free (left) and strong chromatic (right) guarding

For an orthogonal polygon P we construct its induced *visibility arrangement* $\mathcal{A}^r(P)$: For each reflex vertex of P we extend both incident boundary edges into int$P$ until they meet the boundary again, therefore defining a subdivision of the polygon. The 2-dimensional faces of this arrangement are rectangles. Clearly, points from the interior of the same rectangle (subsequently called *cell*) have the same visibility polygon.
Finally, we define special classes of orthogonal polygons. A *histogram* has a boundary *base edge* e connecting two convex vertices such that $V(e) = P$. A histogram that is a star is called a *pyramid*.

Conflict-free and strong chromatic guarding:
A set G of points is a *guard set* for an orthogonal polygon P if their visibility polygons jointly cover the whole polygon. If in addition each guard $g \in G$ is assigned one color $\gamma(g)$ from a fixed finite set of colors $[t] = \{1, 2, \ldots, t\}$ we have a *chromatic guarding* (G, γ). Next we give the central definitions.
A chromatic guard set (G, γ) for P is *strong* if each point in P sees only differently colored guards.
(G, γ) is *conflict-free* if for any point $p \in P$ there is at least one guard in the guard set $G(p) = V(p) \cap G$ whose color is unique.
Figure 1 illustrates both concepts. We denote by $\chi_{cf}(P)$ the minimal t such that there is a conflict-free chromatic guarding set for P using t colors. Maximizing this value over all polygons with n vertices from a specified polygon class is denoted by $\chi_{cf}(n)$.
Consequently, we denote by $\chi_{st}(P)$ the minimal t such that there is strong chromatic guarding set using t colors. Maximizing this value for all polygons with n vertices from a specified polygon class defines the value $\chi_{st}(n)$. Observe that minimizing the guard number is not part of the objective function. However, in our upper bound proofs we use at most a linear number of guards what is best possible.

*Freie Universität Berlin, Inst. für Informatik, 14195 Berlin, Germany; {hoffmann,kriegel,willerma}@mi.fu-berlin.de
†Dept. of Computer Science, University of California, Santa Barbara, USA. suri@cs.ucsb.edu
‡Dept. of Math and Computer Science, TU Eindhoven, The Netherlands;k.a.b.verbeek@tue.nl

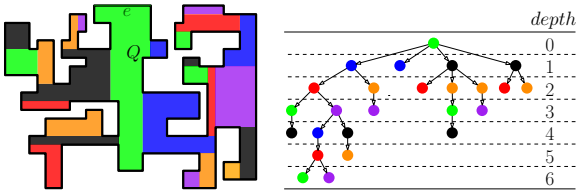


Figure 2: The partition into histograms and the corresponding partition tree.

2 Upper Bounds

We show two upper bounds for simple orthogonal polygons of size n: $\chi_{st}(n) \in O(\log n)$ and $\chi_{cf}(n) \in O(\log \log n)$. These bounds are even realized by guards placed in the interior of visibility cells. This restriction simplifies the arguments and does not affect asymptotic bounds. Furthermore we use the simple fact that a polygon is guarded iff its interior is guarded. The upper bound proof is inspired by ideas developed in [1, 2] for conflict-free guarding of simple polygons based on line visibility. Omitted details can be found in [4].

2.1 Reduction to histograms

We reuse the central concept of independence introduced in [1, 2] for line visibility. Independence means that one can use the same color sets for coloring guards in independent subpolygons. The following definition suffices for our purposes and covers both the strong and the conflict-free variant:

Let P be a simple orthogonal polygon and P_1 and P_2 subpolygons of P. We call P_1 and P_2 *independent* if no point in P can see simultaneously points from $\mathrm{int}P_1$ and $\mathrm{int}P_2$.

Next, we hierarchically subdivide an orthogonal polygon P into a linear number of histograms by a standard window partitioning process, see [1].
The subdivision is represented by a partition tree $\mathcal{H} = \mathcal{H}_P(e)$ with histograms as node set. Let e be a highest horizontal boundary edge. The visibility polygon of e is a histogram. This is the root vertex of $\mathcal{H}$. Now Q splits P into parts and defines a finite set (possibly empty if $Q = P$) of vertical windows $w_1, \ldots, w_k$. Then we recurse, see Figure 2, with the windows being the new base edges. Each window corresponds to a last left or right turn of a shortest orthogonal path from e to the histogram defined by the window. So we can label the histograms left or right.

Let $H_d, d = 0, 1, 2$, be the family of all histograms corresponding to nodes in $\mathcal{H}$ with depth congruent d mod 3. We partition H_d into H_d^L consisting of Q and all those histograms which are left children and, on the other side, H_d^R consisting of the remaining "right" parts. In Figure 2 the six families of histograms are color-coded for illustration. For example, the dark gray histograms are right children with depth congruent 1 mod 3.

Lemma 1 *Let P be a polygon and $H_d^L, d = 0, 1, 2$ the family of histograms corresponding to left nodes in $\mathcal{H}$ with depth congruent d mod 3. Then the interior of histograms in each H_d^L have pairwise link distance at least three, analogously for H_d^R, so they are independent.*

2.2 Guarding a histogram

Consider a histogram $H \subseteq P$ with a top horizontal base edge. We associate with H a tree T as follows. Consider the set of open 2-cells in the visibility arrangement $\mathcal{A}^r$. If several cells have the same visibility polygon we choose the leftmost cell as representative. Let R be the set of all representatives and $B \subseteq R$ the subset of cells adjacent to bottom horizontal edges. We define a partial order for B: We say $b' \leq b$ iff the horizontal polygon edge of b' is not above that of b and there is an $r \in R$ that sees both b and b'. The Hasse diagram of this poset is a tree T which we call the *spine* of H. A *monotone* path π in T is a directed subpath of a root-to-leaf path. It corresponds to a pair (b, b') with $b' \leq b$.

Lemma 2 *There is a bijective mapping Φ between cells of R and monotone paths in T such that two cells are visible from each other iff the corresponding monotone paths in T share a node.*

Proof. Let r be a cell in R. Then $V(r) \cap T$ is some montone path π in T and we set $\Phi(r) = \pi$. For the inverse function let π be a monotone path in T from vertex b down to b'. We associate with π the unique cell $\Phi^{-1}(\pi) = r \in R$ that is vertically aligned with b' and horizontally with b.
We observe that Φ is well-defined by the choice of the leftmost representative for visibility equivalent cells and it is clearly a bijection. Especially, for $\pi = (b, b)$ we have $\Phi^{-1}(\pi) = b$.
For the second assertion consider two cells r, r' visible from each other and the smallest rectangle that includes both. By extending this rectangle downwards it hits a horizontal boundary edge. The vertex of T corresponding to that edge is in both $\Phi(r)$ and $\Phi(r')$. For the other direction consider a cell r'' corresponding to a vertex in $\Phi(r) \cap \Phi(r')$. It has a bottom horizontal edge. We form a rectangle in H above this edge of maximal width and maximal heigth. All cells visible from r'' are in this rectangle, therefore r sees r'. Figure 3 illustrates the bijection. □

Now we translate the geometric concepts of strong and conflict-free guardings of H into equivalent combinatorial questions for the spine T. First of all, a

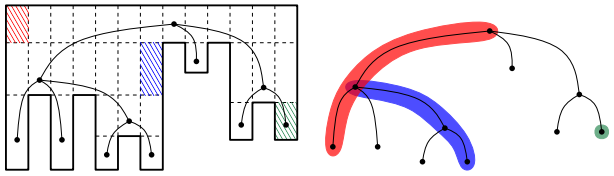


Figure 3: Spine tree and bijection between open cells and monotone paths

colored guard set for H defines a set of colored cells in R and this defines using Φ a covering of T with colored monotone *guarding paths* and vice versa. The condition for strong guarding now reads: No monotone path in T can intersect two guarding paths of the same color.
For conflict-free guardings we have:

Lemma 3 *Colored guards $g_1, \ldots, g_r$ define a conflict-free guarding for H iff for each monotone path π in T there exists a color and exactly one guarding path $\Phi(g_i)$ with that color such that $\pi \cap \Phi(g_i) \neq \emptyset$.*

Proof. Consider the cell $\Phi^{-1}(\pi)$. It is seen by a guard g with a unique color c. Therefore $\Phi(g) \cap \pi \neq \emptyset$. Assume, some other c-colored guarding path $\Phi(g')$ intersects π. Then g' sees $\Phi^{-1}(\pi)$, a contradiction. The other direction is analogous. □

Path compression: We use bottom-up path compression to define a covering (in fact, it is a partition) of T by monotone paths. To this end we form in parallel for all leaves l the maximal length monotone paths $\pi(l)$ that end in l and do not use nodes of outdegree > 1. We cut off all $\pi(l)$ from T. Iterating this procedure yields a unique tree T^*. Its nodes represent monotone paths in T. T^* has depth $O(\log |H|)$ since in each iteration the number of leaves is reduced by at least half. Figure 4 shows an example histogram with its spine tree T. The derived compressed spine is depicted in Figure 5.

Proposition 4 *Let H be a histogram with n vertices. Then there is a strong guarding with $O(\log n)$ colors and a conflict-free guarding with $O(\log \log n)$ colors.*

Proof. We construct the spine T and the compressed tree T^* with depth $O(\log n)$. To get a strong guarding we color the nodes of T^*, i.e. the guarding paths in T, by their depth in T^*.
For a conflict-free guarding consider the color set $[t] = \{1, 2, \ldots, t\}$ and the following recursively defined set of words: $s_1 = 1$ and $s_i = s_{i-1} \circ i \circ s_{i-1}$. Clearly, a prefix of s_t with length k has no more than $\lceil \log(k+1) \rceil$ different colors and each connected subword contains a unique color, [5]. Now we color the nodes of T^* from the root to the leaves with the sequence s_t of length

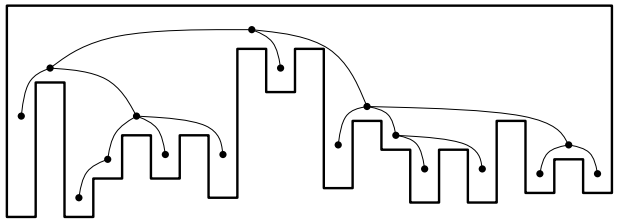


Figure 4: Example histogram with spine tree

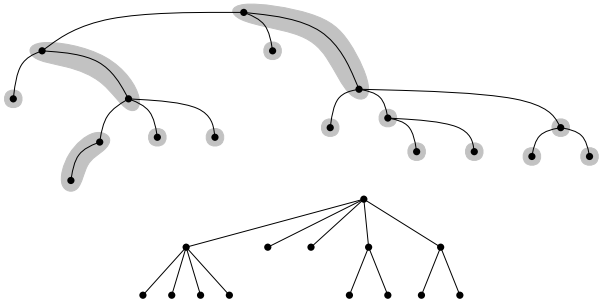


Figure 5: Monotone paths covering the spine and the corresponding compressed spine

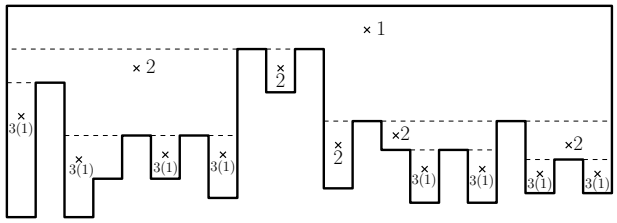


Figure 6: Chromatic guarding positions for the example histogram: with colors {1,2,3} strong, with colors {1,2} conflict-free (in brackets) guarding

at most the height of the tree, that is $O(\log n)$. A color alphabet of size $O(\log \log n)$ suffices. □

We illustrate the construction in Figure 6. Observe that we use the same guard positions for both strong and conflict-free guarding. The compressed spine has depth 2. For the strong guarding we use colors 1,2 and 3 while for the conflict-free version we use the color sequence 1-2-1. The guard positions are the open cells corresponding to the monotone paths via bijection Φ. Moreover, each guard covers a pyramid as indicated in the figure.

Theorem 5 *Let P be an orthogonal polygon with $|P| = n$. We have $\chi_{st}(n) \in O(\log n)$ and $\chi_{cf}(n) \in O(\log \log n)$.*

Proof. We decompose P into 6 families $H_d^L, H_d^R, d = 0, 1, 2$, of pairwise independent histograms each of size at most n. Then we apply Proposition 4. □

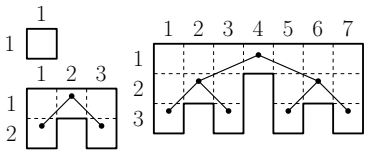


Figure 7: Spike polygons S_1, S_2 and S_3

3 Lower Bounds

All lower bounds established in this paper are based on a simple, recursively defined family of so called spike polygons S_m, where S_1 is a simple square and S_{m+1} is formed by two copies of S_m separated by a vertical spike, but joined by an additional horizontal layer. Figure 7 illustrates this construction together with the subdivision of S_2 into visibility cells and the corresponding spine tree. Obviously, the spine tree T of S_m is a balanced binary tree of height $m-1$ with vertices corresponding one-to-one to bottom cells.
Recall, a colored guard set for S_m corresponds to a covering of T with colored monotone guarding paths and vice versa.

Theorem 6 *For simple orthogonal polygons* $\chi_{st}(n) \in \Omega(\log n)$.

Proof. We show that any strong guarding of S_m requires m different colors. Consider in spine T a guarding path π covering the root with unique color c. Then c does not occur in the left or in the right subtree of the root. By induction we need $m-1$ more colors for the subtree missed by π. Since S_m has $n = 2^{m+1}$ vertices the claim follows. □

Next we analyse the special case that a root-to-leaf path π in T is covered by short guarding paths only. Later we show the existence of such a path.

Lemma 7 *Let* $\mathcal{P} = \{\pi_1, \ldots, \pi_r\}$ *be conflict-free guarding paths for a path* π *with* m *nodes such that* $|\pi_i| = O(m^\epsilon)$ *for some* $0 < \epsilon < 1$ *and* $1 \le i \le r$. *Then this guarding uses at least* $\Omega(\log m)$ *colors.*

Proof. Let $K = max\{|\pi_i|,\ 1 \le i \le r\}$. We subdivide π into $k = \frac{m}{K} \in \Omega(m^{1-\epsilon})$ buckets of size K each. This way every π_i can overlap with at most two buckets. Since $\mathcal{P}$ is induced by a conflict-free guarding, there is a color c_1 such that exactly one π_i (responsible for π) is colored with c_1. Hence there is a subpath of π consisting of at least $\frac{k-2}{2}$ buckets that does not intersect any c_1-colored path. Applying this argument recursively we obtain the following recursive relation for the number of colors needed for k buckets: $T(k) \ge T(\frac{k-2}{2}) + 1$. This recursive relation easily solves to $T(k) \in \Omega(\log k) \subseteq \Omega(\log m^{1-\epsilon}) = \Omega(\log m)$. □

Theorem 8 *For simple orthogonal polygons* $\chi_{cf}(n) \in \Omega(\log\log n)$.

Proof. We start with a conflict-free guarding of $S_m, n = 2^{m+1}$ that uses a minimum number of t colors. By Theorem 5 we have $t \in O(\log\log n) = O(\log m)$. Consider the corresponding family $\mathcal{F}$ of guarding paths in T. We denote by $\mathcal{U}(v_0)$ the set of all guarding paths from $\mathcal{F}$ covering the root v_0 of T with a unique color. Since $|\mathcal{U}(v_0)| \le t$ there must be a vertex v_1 at depth $\lfloor \log t \rfloor + 1$ that is not part of any path from $\mathcal{U}(v_0)$. Now we iterate starting from v_1. We take all guarding paths covering v_1 with a unique color and we determine a node v_2 at depth $2\lfloor \log t \rfloor + 2$ missed by these paths, and so on. We call the v_i's *checkpoints*. The checkpoints define a root-to-leaf path π with length $m = \log n - 1$. Consider $\mathcal{F}_\pi = \{\pi \cap \pi_i | \pi_i \in \mathcal{F}\}$. Now form a new family $\mathcal{U}_\pi$ that consists of all maximal subpaths σ of members $\pi_i \cap \pi \in \mathcal{F}_\pi$ such that σ does not intersect any other member of the same color in $\mathcal{F}_\pi$. Let $\pi' \subset \pi$ and assume $\pi_i \in \mathcal{F}$ gives a unique color to π'. Then $\pi' \cap \pi_i$ is part of some path in $\mathcal{U}_\pi$. Thus $\mathcal{U}_\pi$ is a conflict-free guarding path family for π. By construction, paths in $\mathcal{U}_\pi$ have length at most $2\lfloor \log t \rfloor + 1$. Now we can apply Lemma 7. Since $2\log t + 1 \in O(\log\log m) \subseteq O(m^{0.5})$ we get $t \in \Omega(\log m) = \Omega(\log\log n)$. □

Applying another, more involved combinatorial tableaux technique, we can prove a first nontrivial lower bound of $\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$ for conflict-free guarding of orthogonal polygons with line visibility. These tableaux comprise necessary conditions a color-multiset family must have to define a conflict-free guarding of appropriately vertically stretched spike polygons S_m, see [4] for details.
However, the best known upper bound for this problem setting is $O(\log n)$, as shown in [1]. This upper bound holds also for general simple polygons [2]. Closing these gaps is a challenging open problem.

References

[1] A. Bärtschi, S. Suri. Conflict-free Chromatic Art Gallery Coverage. Algorithmica 68(1): 265–283, 2014.

[2] A. Bärtschi, S.K. Ghosh, M. Mihalak, T. Tschager, and P. Widmayer. Improved bounds for the conflict-free chromatic art gallery problem. In Proc. of 30th SoCG, pages 144–153, 2014.

[3] L. H. Erickson and S. M. LaValle. An Art Gallery Approach to Ensuring that Landmarks are Distinguishable. In Proc. Robotics: Science and Systems VII, Los Angeles, pages 81–88, 2011.

[4] F. Hoffmann, K. Kriegel, M.Willert. Almost Tight Bounds for Conflict-Free Guarding of Orthogonal Art Galleries. arXiv:1412.3984 [cs.CG], 2014.

[5] S. Smorodinski. Conflict-Free Coloring and its Applications. In Geometry - Intuitive, Discrete, and Convex, volume 24 of Bolyai Society Mathematical Studies, Springer Verlag, Berlin 2014.

Special Guards in Chromatic Art Gallery

Hamid Hoorfar* Ali Mohades†

Abstract

We present two new versions of the chromatic art gallery problem that can improve upper bound of the required colors pretty well. In our version, we employ restricted angle guards so that these modern guards can visit α-degree of their surroundings. If α is between 0 and 180 degree, we demonstrate that the strong chromatic guarding number is constant. Then we use orthogonal 90-degree guards for guarding the polygons. We prove that the strong chromatic guarding number with orthogonal guards is the logarithmic order. First, we show that for the special cases of the orthogonal polygon such as snake polygon, staircase polygon and mounts polygon, the number of colors is constant. We decompose the polygon into parts so that the number of the conflicted parts is logarithmic and every part is snake. Next, we explain the chromatic art gallery for the orthogonal polygon with guards that have rectangular visibility. We prove that the strong chromatic guarding number for orthogonal polygon with rectangular guards is logarithmic order, too. We use a partitioning for orthogonal polygon such that every part is a mount, then we show that a tight bound for strong chromatic number with rectangular guards is $\theta(\log n)$

1 Introduction

New approach of the art gallery problem was raised with Erickson and LaValle [2], which maximized the compatible guards so that for two guards whom their intersection of visibility polygons is not empty must be spent a new color as cost. In the other words, the chromatic art gallery find the minimum number of colors that always sufficient and sometimes necessary for guarding the entire polygon. It is called the chromatic guarding number. Let $\chi_G(P)$ denotes the chromatic guarding number of polygon P. We extend this notation so that $\chi_G^{\alpha}(P)$ denotes the chromatic guarding number with α-degree guards, and $\chi_G^{rec}(P)$ denote the chromatic guarding number with rectangular guards, properly. The motivation of offering this problem was in robot controlling with wireless navigators whom we can set the angle of their ranges. In many cases, these navigators have the 90-degree range, and they are orthogonal corresponding to the environment, because of it, we introduce the chromatic art gallery with orthogonal 90-degree guards in the orthogonal polygons. Erickson and LaValle showed that for a spiral polygon, the chromatic guarding number is at most 2 and for a staircase polygon is at most 3, then they showed that for every positive number k, there exists a polygon with $4k$ vertices such that$\chi_G(P_k) \geq k$. Also, they showed that for every odd number$k$ there is an orthogonal polygon with $4k^2 + 10k + 10$ vertices such that $\chi_G(P_k) \geq k$ [2]. We extend these result with α-degree guards so that $\chi_G^{\alpha}(P) \leq 2$ and for orthogonal guards $\chi_G^{O}(P) \in O(\log n)$ wherever $\chi_G^{O}(P)$ denotes the chromatic guarding number with the orthogonal 90-degree guards. After that, we extend these result with rectangular guards so that $\chi_G^{rec}(P) \in \theta(\log n)$

2 Basic Definitions

Let polygon P be a connected simple subset of $\mathbb{R}^2$ with ∂P as its boundary. $p \in P$ is visible from $q \in P$ if the segment pq is a subset of P. For every point p in the polygon, $V(p)$ indicates visibility polygon of p so that $V(p) = \{q \in P \; s.t. \; p \text{ is visible from } q\}$. The guard set S is a finite set of points in the polygon such that $\bigcup_{s \in S} V(s) = P$, every element of S is called guard [1]. A pair of guards s and t is named incompatible whenever $V(s) \cap V(t) \neq \emptyset$. Let $C(s)$ denotes the minimum colors necessary for coloring the guard set S so that every pair of incompatible guards have different colors. Furthermore, let $T(p)$ denote the set of all guard sets in P, and $\chi_G(P) = min_{s \in T(p)} C(s)$ is called the chromatic guarding number. The chromatic art gallery problem minimizes the chromatic guarding number rather the guarding number [3]. Let α-guard denotes the guard whom its visual field is $(v, v + \alpha)$ wherever v is an arbitrary angle and $\alpha \in (0, 180]$, for instance, 90-guard is a guard with visual field equal to $(v, v + 90)$. In addition, let O-guard denotes the orthogonal 90-guard. Suppose $\chi_G^{\alpha}(p)$ denotes the chromatic guarding number with the α-guards, and we extend the notations so that $\chi_G^{O}(p)$ indicates the chromatic guarding number with the O-guards as well. Also:
$V^{\alpha}(g) = \{p \in P \mid q \text{ is visible from } \alpha - guard \; g\}$
$V^{O}(g) = \{q \in P \mid q \text{ is visible from } O - guard \; g\}$
In the orthogonal polygon, a horizontal (vertical)

*Laboratory of Algorithm and Computational Geometry, Faculty of Mathematics and Computer Science, Amirkabir University of Technology, Iran.

†Laboratory of Algorithm and Computational Geometry, Faculty of Mathematics and Computer Science, Amirkabir University of Technology, Iran.

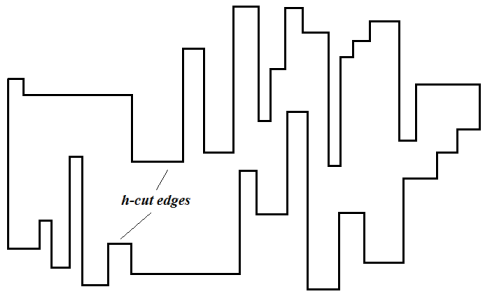


Figure 1: An orthogonal polygon that has no v-cut edge.

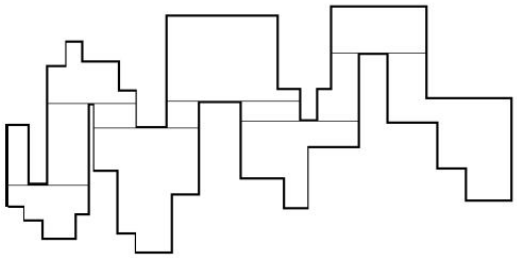


Figure 2: A snake polygon

edge e_h (e_v) that its two end points are reflex vertices is called h-cut edge (v-cut edge) [4]. For any two points p and q in a rectilinear polygon P, if the aligned rectangle with p and q as opposite corners lies totally inside P, then p and q are called rectangularly visible [4]. Also for any point p in polygon P, $V^{rec}(p)$ denotes the rectangular visibility area of P, such that:
$V^{rec}(p) = \{q \in P | p \text{ is rectangularly visible from } q\}$
Every guard can see around it rectangularly is represented by the r-guard. Assume guard set S that is a finite set of r-guards in the polygon so that $\bigcup_{s \in S} V^{rec}(s) = P$ then we call that P is r-visible from S. we extend the use of the notations to r-guards as $\chi_G^{rec}(P) = \min_{S \in T(p)} C(S)$ whenever S is an r-guard set. And assume these notations:
$V^{\perp}(p) = \{q \in P | q \text{ is orthogonally visible from point } p\}$
$V^{\perp}(e) = \{q \in P | q \text{ is orthogonally visible from segment } e\}$

Observation 1 *Every orthogonal polygon that has no h-cut edge (v-cut edge) is y-monotone (x-monotone) polygon. As the Figure 1 shows.*

In the following we define a new special polygon, we call it, the snake polygon.

Definition 1 *A polygon P is called snake polygon if:*
1. P is an orthogonal x-monotone (y-monotone).
2. The line segment created by extending an h-cut edge to the boundaries of ∂P can decompose P into exactly three subpolygons.
3. Each of its sub polygon must be xy-monotone. see Figure 2.

As the sample, every staircase polygon is a snake polygon; similarly, every xy-monotone polygon is a snake polygon as well. We define two special polygons, mounts polygon and mount polygon as following.

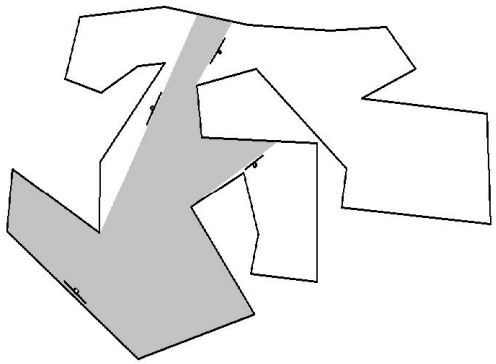


Figure 3: 180-degrees guard

Definition 2 *Polygon P is called mounts polygon if it has at least an edge $e \in P$ such that $V^{\perp}(e) = P$, nominate e, base edge.*

Definition 3 *The mounts polygon P is named mount polygon if P is monotone corresponding to a line perpendicular to its base edge.*

3 A tight bound on the chromatic guarding number of the general polygon with α-guards

Consider a simple polygon P. Select an arbitrary edge e_1 then place a 180-guard g_1 on it. Process $V^{180}(g_1)$ [5], every connected part of its boundary, which is not belonged ∂P is called window. Continually place 180-guards on the created windows of previous step until the entire polygon is covered, see Figure 3

We demonstrated that in the general polygon with 180-degree guards the chromatic guarding number is 1, i.e.$\chi_G^{180}(p) = 1$. Now, we can replace every 180-guard with at most two colors of α-guards. We want $\lfloor \frac{180}{\alpha} \rfloor$ guards in the same color and perhaps one guard

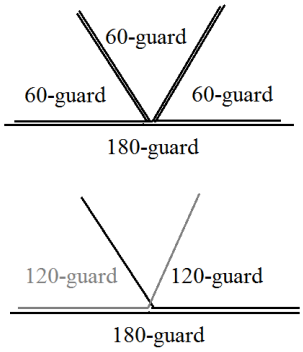


Figure 4: Replace a 180-degrees guard with α-guards

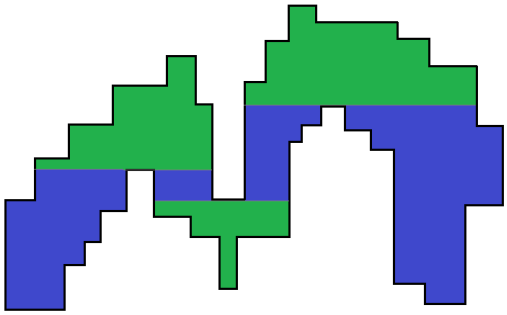


Figure 5: O-guarding a snake polygon

in the different color. Wherever $\frac{180}{\alpha}$ is not an integer such that it is shown in Figure 4. Hence, if $0 < \alpha \leq 180$, then $\chi_G^{\alpha}(p) \leq 2$.

4 Some special polygons with O-guards

Suppose that P is a snake polygon, we present a partitioning so that its chromatic guarding number will be constant. Extend all h-cut edges in the snake polygon, because of this, the polygon decomposes to sub polygons such that all of them are staircase or mount polygons. We can guard all these staircase parts with one color of the O-guard independently, see Figure 5. The green parts are the mount polygons, and the blue ones are staircase polygons.

Observation 2 *For every staircase polygon P when the bottom edge is not seen, $\chi_G^O(P) = 1$.*

Observation 3 *For every mount polygon P when the bottom edge is not seen, $\chi_G^O(P) = 1$, see Figure 6.*

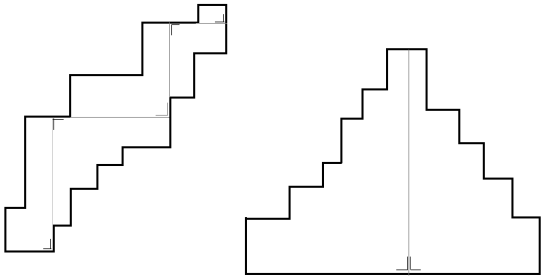


Figure 6: The staircase polygon and mount polygon can be guarding with one color of O-guards

If the bottom edge is not seen, then the visibility polygon of the O-guards in two different parts will not be incompatible. We place the green O-guards in the green areas and the blue ones in the blue areas. Therefore, we can cover the entire snake polygon with at most two colors. So the chromatic guarding number of the snake polygon is less than or equal 2.

Observation 4 *The chromatic guarding number of the snake polygon with O-guards is at most 2.*

5 A tight upper bound on the chromatic guarding number of the orthogonal polygon with O-guards

In this section, we present a partitioning for the orthogonal polygon such that every part is a snake polygon. We construct two partitions in two orientations both horizontally and vertically. Call horizontal (vertical) partitioning h-cutting (v-cutting), then we nominate duality graph of h-cutting (v-cutting), h-tree (v-tree). Extend all h-cut (v-cut) edges in the polygon until intersect the boundary, start from the lowest (leftmost) part, the duality node corresponding to it must be root, then draw a directed edge from the root to all its neighbors, continue it recursively until the h-tree (v-tree) is built. See Figure 7.

We modify any paths in the h-tree, so that duality of them will be snake polygons. For this purpose, we cut the v-cut edges occur in the path. We know that these removed parts will cover with guarding in the other orientation partitioning, certainly. See Figure 8.

Similarly, consider this process for v-tree. If we find the chromatic guarding number to cover the h-tree, double of this number will be the chromatic guarding number for the entire orthogonal polygon. We know that every modified path in the duality tree has the chromatic guarding number at most 2 with the condition that the bottom edge is not seen.

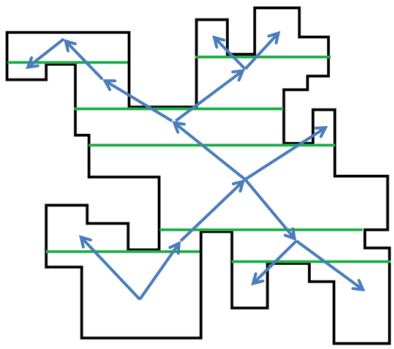


Figure 7: Every path in the duality graph decomposes to a snake polygon.

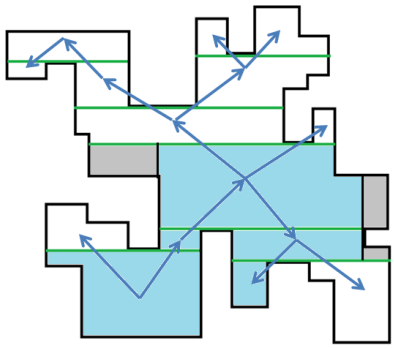


Figure 8: Partitioning of an orthogonal polygon.

Follow this algorithm:
1. Find the path from the root to the leaf in every remained tree so that with removing it, the tree decomposes into at least two sub trees which their size are at most half of the total tree.
2. Guard all paths from previous step with two new colors.
3. Remove the path from the total tree and remain the sub tree(s).
4. If all remained sub trees are not empty, then go to step 1.

Lemma 1 *The number of iterations of the algorithm is* $O(\log n)$.

Proof. In iteration the size of the problem will be at most half of the previous step then we can write: $f(n) \leq f(\frac{n}{2}) + 2 \Rightarrow f(n) \in O(\log n)$, which $f(n)$ means the complexity of problem. □

Theorem 2 *The chromatic guarding number of the orthogonal polygon with O-guards is the logarithmic order.*

Proof. Using Lemma 1, we know that for guarding the h-cutting, we need $O(\log n)$ colors of guards, by this guarding, some parts of the polygon is not covered, because of the modified paths, that cut in the cross orientation. Nevertheless, we are guarding these remained parts during guarding v-cutting, completely. For it, we pay $O(\log n)$ colors of guards as cost. As a result, the chromatic guarding number for the orthogonal polygon belongs to $O(\log n)$. □

6 An upper bound on the chromatic guarding number of the general polygon with O-guards

We present an algorithm for the general polygon such that we remove an orthogonal polygon from it. By it, all remained parts are spiral polygons or triangles so that we can guard them with at most two colors compatibly. Follow this algorithm:
1. Draw a horizontal line from every vertex in the polygon.
2. Draw a vertical line from every intersection of lines and polygon.
3. Select rectangles that occurs in the polygon completely and make an orthogonal polygon.
4. Consider remained parts, we can guard all disjoint polygon compatibly.

Observation 5 *Each remained part is the spiral polygon so that reflex chain of it, is orthogonal. (Triangle is a special type of the spiral polygon).*

Observation 6 *The chromatic guarding number for the spiral polygon with O-guards in which the reflex chain is not seen is at most 2.*

The condition that the reflex chain doesnt must be seen appears because, the spiral parts can be guarded independently.

Lemma 3 *The chromatic guarding number for the general polygon with O-guards is* $O(\log n)$.

Proof. Decompose a general polygon into orthogonal polygons that its vertices can be $O(n^2)$, nevertheless the $\chi_G^O(P) \in O(\log n)$, for remained spiral polygon $\chi_G^O(P) \leq 2$, therefore, for the general polygon P, we have: $\chi_G^O(P) \in O(\log n)$

□

7 A tight bound on the rectangular chromatic guarding number of the orthogonal polygon

Observation 7 *The rectangular chromatic guarding number of the mounts polygon belongs to* $\theta(\log n)$.

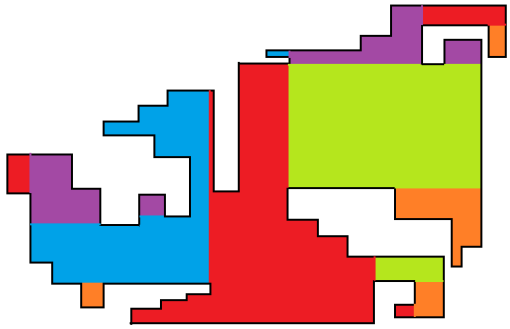


Figure 9: Partitioning the orthogonal polygon to the mounts polygons.

We found that the rectangular chromatic guarding number of mounts polygon is the logarithmic order. Now, we want to extend this result for all orthogonal polygons as well. First, we divide the polygon into the mounts polygons. Then, the lowest edge of the polygon, nominate e and process $V^{\perp}(e)$. Suppose the boundary of $V^{\perp}(e)$, every connected part of the boundary which is not part of the boundary of the total polygon is called window.

Observation 8 *The connected window in $V^{\perp}(e)$ is line segment.*

Now, find $V^{\perp}(e)$ for every obtained window since the entire polygon is supported. See Figure 9.

We demonstrate that if we can cover the mounts polygon with $\log n$ colors, then we can cover all parts of the partitioning with $5 \log n$ colors, in other words: $\chi_G^{rec}(P_{orthogonal}) \in \theta(\log n)$.
If a mounts polygon part has the window in its boundary, then it has incompatibility with other parts. The incompatibility area is a rectangle such that its width is equal to width of the window. Start from first part, color it with the first color and color all parts that immediately left of it (call Left-Parts) with the second color and all parts that immediately right of it (call Right-Parts) with the third color. So, color all parts top of Left-Parts and Right-Parts with 4th color and all parts bottom of Left-Parts and Right-Parts with 5th color. Any parts in the same color want the same log n colors of r-guards. By this strategy, we can color the remained parts with the same five colors so that showed in Figure 9.

Observation 9 *The same colored parts are compatible.*

Theorem 4 *The rectangular chromatic guarding number that always sufficient and sometimes necessary for covering an orthogonal polygon is $O(\log n)$.*

Proof. We found that there is an orthogonal polygon which is needed $\theta(\log n)$ as its rectangular chromatic guarding number, and using above method are shown that $\theta(\log n)$ is sufficient for all orthogonal polygons, so the proof is completed. □

8 Conclusion

In this paper, we explained the chromatic art gallery on the simple polygon with 90-guards that its visibility field of them is one of the $(0, 90)$, $(90, 180)$, $(180, 270)$ or $(270, 360)$. This type of guards named O-guard. In many cases in real-world, the guards can see a limited angle of its around. This motivated that we define the new version of the chromatic art gallery problem so that increase the conflict-free guards from lower bound $\frac{n}{4}$ to upper bound $O(\log n)$. Then, we explained the chromatic art gallery for the orthogonal polygon with a special type of guards are called r-guards. This type of guards has r-visibility such that any two points p and q in a rectilinear polygon P are called r-visible, if the aligned rectangle with p and q as opposite corners lies totally inside P.

References

[1] J. ORourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, Cambridge, 1987, UK.

[2] L. H. Erickson. S. M. LaValle. *An art gallery approach to ensuring that landmarks are distinguishable.* The Journal of Science and Systems, 2011, Los Angeles, USA.

[3] L. H. Erickson. S. M. LaValle. *A chromatic art gallery problem.* Technical report, 2010, illinois, USA.

[4] S. K. Ghosh. *Visibility Algorithms in the plane.* Cambridge University Press, 2007, UK.

[5] H. ElGindy. D. Avis. *A Linear algorithm for computing the visibility polygon from a point.* The journal of Algorithms, 1987.

Column Planarity and Partial Simultaneous Geometric Embedding for Outerplanar Graphs*

Luis Barba† Michael Hoffmann‡ Vincent Kusters‡

Abstract

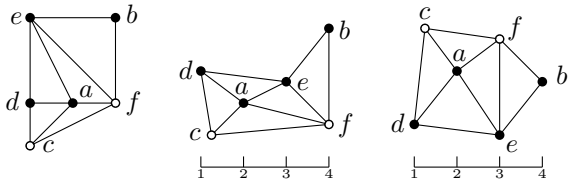
Given a graph $G = (V, E)$, a set $R \subseteq V$ is *column planar* in G if we can assign x-coordinates to the vertices in R such that every assignment of y-coordinates to R gives a partial embedding of G that can be completed to a plane straight-line embedding of the whole graph. This notion is strongly related to unlabeled level planarity. We prove that every outerplanar graph on n vertices contains a column planar set of size at least $n/2$.

We use this result to show that every pair of outerplanar graphs G_1 and G_2 on the same set V of n vertices admit an $(n/4)$-*partial simultaneous geometric embedding* (PSGE): a plane straight-line embedding of G_1 and a plane straight-line embedding of G_2 such that $n/4$ vertices are mapped to the same point in the two drawings. This is a relaxation of the well-studied notion of *simultaneous geometric embedding*, which is equivalent to n-PSGE.

1 Introduction

The notion of *column planarity* was originally introduced by Evans et al. [6]. Informally, given a graph $G = (V, E)$, a set $R \subseteq V$ is *column planar* in G if we can assign x-coordinates to the vertices in R such that any assignment of y-coordinates to R gives a partial embedding of G that can be completed to a plane straight-line embedding of the whole graph. More formally, R is column planar in G if there exists an injection $\rho : R \to \mathbb{R}$ such that for all ρ-*compatible* injections $\gamma : R \to \mathbb{R}$, there exists a plane straight-line embedding of G where each $v \in R$ is embedded at $(\rho(v), \gamma(v))$. Injection γ is ρ-compatible if the combination of ρ and γ does not embed three vertices on a line. See Figure 1.

Column planarity is both a generalization and a strengthening of *unlabeled level planarity (ULP)*. A graph $G = (V, E)$ is ULP if for all injections $\gamma : V \to \mathbb{R}$, there exists an injection $\rho : V \to \mathbb{R}$, so that embedding each $v \in V$ at $(\rho(v), \gamma(v))$ results in a plane straight-line embedding of G. If V is column planar in G, then G is ULP. Estrella-Balderrama, Fowler and Kobourov [5] introduced ULP graphs and characterized ULP trees in terms of forbidden subgraphs. Fowler and Kobourov [7] extended this characterization to general ULP graphs. ULP graphs are exactly the graphs that admit a simultaneous geometric embedding with a monotone path: this was the original motivation for studying them.



(a) (b) (c)

Figure 1: A graph with column planar set $R = \{a, b, d, e\}$ and $\rho = \{a \mapsto 2, b \mapsto 4, d \mapsto 1, e \mapsto 3\}$. (b-c) depict completed embeddings for two different assignments $\gamma : R \to \mathbb{R}$.

Following the characterization of ULP graphs, Di Giacomo et al. [4] introduce a family of graphs called *fat caterpillars* and prove that they are exactly the graphs $G = (V, E)$ where V is column planar in G (they call such graphs EAP graphs). Evans et al. [6] prove near-tight bounds for column planar subsets of trees: any tree on n vertices contains a column planar set of size at least $14n/17$ and for any $\epsilon > 0$ and any sufficiently large n, there exists an n-vertex tree in which every column planar subset has size at most $(5/6 + \epsilon)n$. Furthermore, they show that outerpaths (outerplanar graphs whose weak dual is a path) always contain a column planar subset of size at least $n/2$. In this paper, we prove that this bound holds for general outerplanar graphs.

Evans et al. [6] apply their results on column planarity to give bounds for *k-partial simultaneous geometric embedding (k-PSGE)*. This problem is a relaxation of *simultaneous geometric embedding (SGE)*, which was introduced by Brass et al. [3]. Given graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ on the same set of n vertices, an SGE of G_1 and G_2 is a pair of plane straight-line embeddings $\varphi_1 : V \to \mathbb{R}^2$ of G_1 and

*M. Hoffmann and V. Kusters are partially supported by the ESF EUROCORES programme EuroGIGA, CRP GraDR and the Swiss National Science Foundation, SNF Project 20GG21-134306.

†School of Computer Science, Carleton University, Ottawa, Canada. Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium. lbarbafl@ulb.ac.be

‡Department of Computer Science, ETH Zurich, Switzerland.

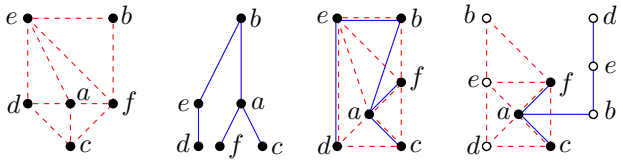


Figure 2: (a-b) Graphs G_1 and G_2 on the same vertex set. (c) An SGE of G_1 and G_2. (d) A 3-PSGE of G_1 and G_2.

$\varphi_2 : V \to \mathbb{R}^2$ of G_2 such that $\varphi_1 = \varphi_2$. See Figure 2c. Conversely, in a k-PSGE of G_1 and G_2, we require $\varphi_1(v) = \varphi_2(v)$ only for some k vertices in V. More formally, a k-PSGE of G_1 and G_2 is a pair of injections $\varphi_1 : V \to \mathbb{R}^2$ and $\varphi_2 : V \to \mathbb{R}^2$ such that (i) the straight-line drawings $\varphi_1(G_1)$ and $\varphi_2(G_2)$ are both plane; (ii) if $\varphi_1(v_1) = \varphi_2(v_2)$ then $v_1 = v_2$; and (iii) $\varphi_1(v) = \varphi_2(v)$ for at least k vertices $v \in V$ [6]. See Figure 2d. An n-PSGE is simply an SGE.

Brass et al. [3] show that two paths, cycles, or caterpillars always admit an SGE. On the negative side, they prove that two outerplanar graphs or three paths sometimes do not admit an SGE. Bläsius et al. [2] give an excellent survey of the subsequent papers on simultaneous embeddings. We highlight the negative result by Geyer et al. [8] that there exist two trees that do not admit an SGE and the result by Angelini et al. [1] that there exist a tree and a path that do not admit an SGE. These negative results motivated the study of PSGE. Evans et al. [6] show that if a set R is column planar in both G_1 and G_2, then G_1 and G_2 admit a $|R|$-PSGE. Di Giacomo et al. [4] independently prove this for $R = V$. Combining their lower bounds on the size of column planar sets with a pigeonhole argument, Evans et al. show that every two trees admit a (11/17)-PSGE.

A result from Goaoc et al. [9] on the untangling of outerplanar graphs, implies that any two outerplanar graphs G_1 and G_2 on n vertices admit a $\sqrt{n/2}$-PSGE.

In this paper, we prove that every outerplanar graph contains a column planar set of size at least $n/2$. We then use this result to show that every two outerplanar graphs on n vertices admit an $(n/4)$-PSGE.

1.1 Outline

We first give an outline of our approach. Consider an outerplanar graph G on n vertices. We first define the *chord graph* of G, which contains only the "long" chords of the graph. We show that the chord graph has an independent set $\mathcal{I}$ of size at least $\frac{n+2}{2}$. We show that $\mathcal{I}$ is almost column planar in G: it suffices to remove at most one vertex. This gives a column planar set of size at least $n/2$ in G.

For our second result, consider two outerplanar graphs G_1 and G_2 on the same set of n vertices. It suffices to compute a set R with $|R| \geq n/4$ that is column planar in both G_1 and G_2. The result of Evans et al. [6] implies then that G_1 and G_2 admit a $|R|$-PSGE. We first compute a column planar set R_1 in G_1. Next, we compute a column planar set R in $G_2[R_1]$ with $|R| \geq n/4$. Since $R \subseteq R_1$, the set R is column planar in both G_1 and G_2, and hence the statement follows.

2 Column Planarity in Outerplanar Graphs

In this section we show that every outerplanar graph has a column planar subset containing at least half of its vertices. Let $G = (V, E)$ be an outerplanar graph with n vertices. Assume without loss of generality that G is maximal outerplanar.

Let $v_0, v_1, \ldots, v_{n-1}$ be the sequence of vertices of V along the unique Hamiltonian cycle of G. Consider the following *removal procedure*: Choose an arbitrary vertex of G of degree two different from v_0 and v_{n-1}, remove it from the graph and repeat recursively. Since every maximal outerplanar graph has two nonadjacent vertices of degree 2, and since removing such a vertex maintains maximal outerplanarity, such a vertex always exists. The *removal order* of the the vertices $V \setminus \{v_0, v_{n-1}\}$ is the order in which they are removed by this procedure. For $0 \leq i < n$, let

$$V(v_i) = \{v_j \in V : v_j \text{ was removed before } v_i\}.$$

Let $N^+(v_i)$ be the closed neighborhood of v_i. For $0 < i < n-1$, the *left index* ℓ_i of v_i is the smallest index such that $v_{\ell_i} \in N^+(v_i)$. Similarly, the *right index* r_i of v_i is the largest index with $v_{r_i} \in N^+(v_i)$. Naturally, $v_{\ell_i} \leq v_i \leq v_{r_i}$.

Lemma 1 *Let v_i be a vertex with $0 < i < n-1$ and suppose that there is a vertex v_j with $i \neq j$ and $\ell_i < j < r_i$. Then all neighbors of v_j are in $V(v_i)$.*

Proof. Let $\ell = \ell_i$ and $r = r_i$ and assume without loss of generality that $i < j$. Since $i < r-1$, the edge $v_i v_r$ is a chord of G. See Figure 3. Hence, the removal of v_i and v_r splits G into two connected components H_1 and H_2 such that $v_j \in H_1$ and $v_0 \in H_2$. Note that v_j neighbors no vertex in H_2. We claim that all the vertices in $V \setminus V(v_i)$ lie in H_2. If this claim is true, then v_j neighbors no point in $V \setminus V(v_i)$, which proves the statement.

Assume for a contradiction that there is a vertex $v \in V \setminus V(v_i)$ that belongs to H_1. Therefore, v lies after v_i in the removal order. Since (i) there is no edge between a vertex of H_1 and a vertex of H_2, (ii) H_1 contains a vertex after removing v_i (namely v), and (iii) H_2 contains a vertex after removing v_i (namely v_0), the graph $G[V \setminus V(v_i)]$ induced by $V \setminus V(v_i)$ is disconnected. However, the removal procedure described above only removes ears of the graph and cannot disconnect it—a contradiction that comes from assuming that v belongs to H_1. □

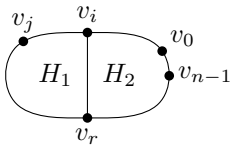


Figure 3: Division into connected components in the proof of Lemma 1.

Let $E_C \subset E$ be the set of all chords of G having endpoints whose removal splits G into components with at least 2 vertices. That is, the chords adjacent to ears of G are not part of E_C; see Figure 4. Let $C = (V, E_C)$ be the *chord graph* of G.

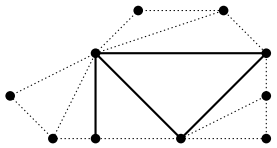


Figure 4: An outerplanar graph $G = (V, E)$. The edge set E_C is drawn solid; the other edges are dotted.

Lemma 2 *Let $\mathcal{I} \subset V$ be an independent set of C such that there is an edge of the Hamiltonian cycle of G whose endpoints are both not in $\mathcal{I}$. Then $\mathcal{I}$ is column planar in G.*

Proof. Let $v_0, v_1, \ldots, v_{n-1}$ be the sequence of vertices of V along the unique Hamiltonian cycle of G such that v_0 and v_{n-1} are not in $\mathcal{I}$. To set the x-coordinate of the vertices in $\mathcal{I}$, we define the injection $\rho : \mathcal{I} \to \mathbb{R}$ such that $\rho(v_i) = i$.

For any ρ-compatible injection $\gamma : \mathcal{I} \to \mathbb{R}$, we need to show that there exists a plane straight-line embedding of G where each $v_i \in \mathcal{I}$ is embedded at $\varphi(v_i) = (\rho(v_i), \gamma(v_i))$.

We first show that φ is a plane straight-line embedding of the graph $G[\mathcal{I}]$. Since $\mathcal{I}$ is an independent set of C, we know that if two vertices $v_i, v_j \in \mathcal{I}$ are adjacent in G such that $i < j$, then either $j = i + 1$ or $j = i + 2$. Otherwise, the removal of v_i and v_j splits G into two graphs, each with at least two vertices, which implies that the edge v_iv_j belongs to C: a contradiction. Furthermore, if $\{v_i, v_{i+2}\} \in E$ then the neighbours of v_{i+1} are exactly v_i and v_{i+2}. Therefore, φ is a plane straight-line embedding of $G[\mathcal{I}]$.

We now describe an algorithm that places the remaining vertices of V to obtain a plane straight-line embedding of G. The algorithm is incremental and adds one vertex at the time in the order given by the removal order.

Let X_i be the set of vertices that have already been placed, starting with, $X_0 = \mathcal{I}$. We never embed two vertices at the same x-coordinate. We say that the *visibility invariant* holds if each vertex of X_i that neighbors a vertex of $V \setminus X_i$ in G is *visible from below*, i.e., the ray shooting downwards from this vertex intersects no edge of the embedding of $G[X_i]$. We can see that the visibility invariant holds for X_0 as follows. Suppose that there is a vertex v_k that is not visible from below. Then the ray from v_k downward intersects some edge $\{v_i, v_j\}$. Since v_i and v_j are independent in C and since $i < k < j$, we must have $k = i + 1$ and $j = i + 2$. But then the only neighbors of v_k are v_i and v_j, and hence v_k does not neighbor a vertex of $V \setminus X_0$, as required.

For any $i \geq 0$, let v_j be the first vertex in $V \setminus X_i$ according to the removal order and let $X_{i+1} = X_i \cup \{v_j\}$. Let $\ell = \ell_j$ and $r = r_j$ be the left and right indices of v_j, respectively.

We place v_j at coordinates (j, y_j), where y_j is a sufficiently small number such that all neighbors of v_j in X_i are visible from v_j. This number always exist by the visibility invariant and since we never embed two vertices with the same x-coordinate. Because $\ell \leq j \leq r$ by Lemma 1, only vertices strictly between v_ℓ and v_r in the x-order can become not visible from below. However, since $V(v_i) \subset X_{i+1}$, Lemma 1 implies that for every $\ell < k < r$, all neighbors of v_k are in X_{i+1}. Therefore, the visibility invariant is preserved for X_{i+1}.

After this process completes, the only remaining vertices to embed are v_0 and v_{n-1}. Embed v_0 at $x = 0$ and v_{n-1} at $x = n - 1$. Move both down sufficiently far so that the edge $\{v_0, v_{n-1}\}$ does not intersect the drawing so far and so that v_0 and v_{n-1} can both see their neighbors from below. This completes the plane straight-line embedding of G. □

Lemma 3 *The graph C has an independent set of size at least $\frac{n+2}{2}$.*

Proof. Let $\overline{G}$ be the weak dual graph of the complete outerplanar graph G. Let x_i be the number of vertices of degree i in $\overline{G}$. Notice that $\overline{G}$ is a binary tree whose leaves correspond to ears of G. Since the the degree two vertex of an ear in G becomes an isolated vertex in C, we know that C has at least x_1 isolated vertices. Since $\overline{G}$ is a binary tree, we know that $x_1 = x_3 + 2$.

We describe a greedy procedure to construct an independent set $\mathcal{I}$ of C. The algorithm chooses the vertex of smallest degree in the current graph (initially C), adds it to $\mathcal{I}$, and removes its neighbors from the graph. Clearly this procedure generates an independent set. We claim that that $|\mathcal{I}| \geq \frac{n+2}{2}$.

Because C is outerplanar, it is 2-degenerate. Therefore, whenever we add a vertex to $\mathcal{I}$, it has degree 0, 1, or 2. Let n_i be the number of vertices in $\mathcal{I}$ that had degree i at the moment they were chosen. Thus, $|\mathcal{I}| = n_0 + n_1 + n_2$. Moreover, we know that $n_0 \geq x_1$ as isolated vertices of C will be added to $\mathcal{I}$ before any other vertex of C. Thus, $n_0 \geq x_1 = x_3 + 2$.

Let m be the number of bounded faces of C. Since $m \leq x_3$, we conclude that $m + 2 \leq n_0$.

Since removing vertices of degree zero or one does not change the number of bounded faces, we remove a bounded face of the current graph exactly when we add a vertex of degree 2 to $\mathcal{I}$. Thus, $m = n_2$. Therefore, $n_2 \leq n_0 - 2$.

Since every time our algorithm chooses a vertex of degree i we remove its i neighbors from the graph, and since only vertices of degree 0, 1 or 2 are chosen, we conclude that $n = n_0 + 2n_1 + 3n_2$. Because $|\mathcal{I}| = n_0 + n_1 + n_2$, we infer that

$$n = n_0 + 2n_1 + 3n_2 \leq 2(n_0 + n_1 + n_2) - 2 = 2|\mathcal{I}| - 2.$$

Consequently $|\mathcal{I}| \geq \frac{n+2}{2}$. □

If the independent set $\mathcal{I}$ guaranteed by Lemma 3 does not satisfy the condition of Lemma 2, for instance when n is even and $\mathcal{I}$ is the set of vertices with an even index, then take any $v_i \in V \setminus \mathcal{I}$ and remove v_{i+1} from $\mathcal{I}$. Since the modified $\mathcal{I}$ satisfies Lemma 2, we have

Theorem 4 *Every outerplanar graph on n vertices contains a column planar set of size at least $n/2$.*

3 Application to Partial Simultaneous Geometric Embedding

Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, both on the same set V of n vertices. Let $R_1 \subseteq V$ be column planar in G_1 and let $R_2 \subseteq V$ be column planar in G_2. Evans et al. [6] proved that then G_1 and G_2 admit an $|R|$-PSGE where $R = R_1 \cap R_2$.

For outerplanar graphs G_1 and G_2, let C_1 and C_2 be their chord graphs, respectively. First use Lemma 3 compute an independent set $\mathcal{I}_1$ of size at least $n/2+1$ in C_1. Remove at most one vertex from $\mathcal{I}_1$ to obtain a set R_1 of size at least $n/2$ that is column planar in G_1 by Lemma 2. Next, use Lemma 3 to compute an independent set $\mathcal{I}_2$ of size at least $n/4 + 1$ in the chord graph of $G_2[R_1]$ (after adding edges to make $G_2[R_1]$ maximal outerplanar). Note that $\mathcal{I}_2$ is also independent in C_2, and hence we can remove at most one vertex from $\mathcal{I}_2$ to obtain a set $R \subseteq R_1 \subseteq V$ of size at least $n/4$ that is column planar in G_2 using Lemma 2. Note that R is also column planar in G_1 since $R \subseteq R_1$. Combining this with the aforementioned result of Evans et al. [6] gives our second result.

Theorem 5 *Every two outerplanar graphs on a set of n vertices admit an $(n/4)$-PSGE.*

References

[1] P. Angelini, M. Geyer, M. Kaufmann, and D. Neuwirth. On a tree and a path with no geometric simultaneous embedding. In *Graph Drawing*, pages 38–49. Springer, 2011.

[2] T. Bläsius, S. G. Kobourov, and I. Rutter. Simultaneous embedding of planar graphs. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2013.

[3] P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. P. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007.

[4] E. Di Giacomo, W. Didimo, G. Liotta, H. Meijer, and S. Wismath. Planar and quasi planar simultaneous geometric embedding. In C. Duncan and A. Symvonis, editors, *Graph Drawing*, volume 8871 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2014.

[5] A. Estrella-Balderrama, J. J. Fowler, and S. G. Kobourov. Characterization of unlabeled level planar trees. *Computational Geometry*, 42(6):704–721, 2009.

[6] W. Evans, V. Kusters, M. Saumell, and B. Speckmann. Column planarity and partial simultaneous geometric embedding. In C. Duncan and A. Symvonis, editors, *Graph Drawing*, volume 8871 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2014. To appear at Graph Drawing 2014.

[7] J. J. Fowler and S. G. Kobourov. Characterization of unlabeled level planar graphs. In *Graph drawing*, pages 37–49. Springer, 2008.

[8] M. Geyer, M. Kaufmann, and I. Vrt'o. Two trees which are self-intersecting when drawn simultaneously. *Discrete Mathematics*, 309(7):1909–1916, 2009.

[9] X. Goaoc, J. Kratochvíl, Y. Okamoto, C.-S. Shin, A. Spillner, and A. Wolff. Untangling a planar graph. *Discrete & Computational Geometry*, 42(4):542–569, 2009.

All Good Drawings of Small Complete Graphs*

Bernardo M. Ábrego† Oswin Aichholzer‡ Silvia Fernández-Merchant† Thomas Hackl‡
Jürgen Pammer§ Alexander Pilz‡ Pedro Ramos¶ Gelasio Salazar‖ Birgit Vogtenhuber‡

Abstract

Good drawings (also known as *simple topological graphs*) are drawings of graphs such that any two edges intersect at most once. Such drawings have attracted attention as generalizations of geometric graphs, in connection with the crossing number, and as data structures in their own right. We are in particular interested in good drawings of the complete graph. In this extended abstract, we describe our techniques for generating all different weak isomorphism classes of good drawings of the complete graph for up to nine vertices. In addition, all isomorphism classes were enumerated. As an application of the obtained data, we present several existential and extremal properties of these drawings.

1 Introduction

We consider drawings of simple graphs in the plane or, equivalently, on the sphere. Vertices are represented by distinct points. Edges are drawn as Jordan arcs connecting two vertices (of that edge) and not containing any vertex except those at their endpoints. Note that we do not distinguish between the elements of the graph and their representation in the drawing. A *good drawing* is a drawing of a graph such that any two edges intersect at most once, either at a common endpoint or at a proper crossing, and no three edges cross at a common point. Good drawings have been extensively studied, and are also referred to as "topological graphs" (e.g., in [14]), "simple topological graphs" (e.g., in [9]), or simply "drawings" (e.g., in [8]). We are interested in good drawings of the complete graph K_n on n vertices.

One main motivation for considering good drawings comes from the problem of minimizing the number of crossings in drawings of K_n (where crossings are counted by the overall sum of the number of points in which each pair of edges crosses, as opposed to the number of crossing edge pairs; see [15]). Indeed, for any drawing of a graph, there exists a good drawing of the same graph with at most the same number of crossings. The Harary-Hill conjecture states that the number of crossings in any drawing of K_n is at least $H(n) = \frac{1}{4}\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor\lfloor\frac{n-2}{2}\rfloor\lfloor\frac{n-3}{2}\rfloor$. This has been verified for $n \leq 12$; see [18]. While it has recently been shown that the Harary-Hill conjecture holds for many classes of drawings of K_n (see [1] and references therein), it still remains open for the general case.

Two drawings are *isomorphic* if there is a homeomorphism of the sphere that transforms one drawing into the other. For good drawings, this partitions the infinite number of drawings into a finite number of isomorphism classes; Kynčl [9] showed that this number is in $2^{\Theta(n^4)}$. With applications like determining the crossing number in mind, the following coarser classification turns out to be useful. Two good drawings are *weakly isomorphic* if there is an incidence-preserving bijection between the drawings such that two edges cross in one drawing if and only if their images in the other drawing cross as well. Roughly speaking, weakly isomorphic drawings that are non-isomorphic differ in the order in which their edges intersect; see [4] for details. The number of weak isomorphism classes of K_n is in $2^{n^2\alpha(n)^{O(1)}}$ [11] and $2^{\Omega(n^2)}$ [16].

Already in 1988, Rafla [20] enumerated all weak isomorphism classes of good drawings of K_n for $n \leq 7$ by a computer program, under the (still unproven) assumption that every good drawing contains a simple (i.e., crossing-free) Hamiltonian cycle. Gronau and Harborth [5] enumerated all non-isomorphic good drawings for $n=6$. Here, we describe our construction of all weak isomorphism classes and the enumeration of all isomorphism classes of good drawings of K_n for $n \leq 9$. The resulting data has been used to obtain exact values for various extremal and existential problems on good drawings of K_n, both for $n \leq 9$ and, via extension of relevant instances, for more vertices. Similar data has been successfully used for combina-

*O.A., A.P., and B.V. are supported by the ESF EUROCORES programme EuroGIGA–ComPoSe, Austrian Science Fund (FWF): I 648-N18 and grant EUI-EURC-2011-4306. S.F.-M. is supported by the NSF grant DMS-1400653. T.H. is supported by the Austrian Science Fund (FWF): P23629-N18 'Combinatorial Problems on Geometric Graphs'.

†Department of Mathematics, California State University, Northridge, [bernardo.abrego|silvia.fernandez]@csun.edu

‡Institute for Software Technology, Graz University of Technology, Austria, [oaich|thackl|apilz|bvogt]@ist.tugraz.at

§Institute for Software Technology, Graz University of Technology, Austria, juergen.pammer@alumni.tugraz.at

¶Departamento de Física y Matemáticas, Universidad de Alcalá, pedro.ramos@uah.es

‖Instituto de Fisica, Universidad Autonoma de San Luis Potosi, gsalazar@ifisica.uaslp.mx

torially different configurations of points [2], to obtain counterexamples, induction bases, or, in general, a better intuition for various problems.

In contrast to, e.g., [20], our generation of all weak isomorphism classes is based on rotation systems. In Section 2, we give the basic theoretical background on rotation systems and sketch techniques that reduced the required computational effort. In Section 3, we describe the enumeration of all non-isomorphic drawings of each weak isomorphism class. Applications and the outcome of several computations on the data are given in Section 4. Parts of this work have been presented in the master's thesis [17] of Pammer.

2 Rotation Systems

Rotation systems were devised as tools for investigating embeddings of graphs on higher-genus surfaces [6]. Let D be an (arbitrary) drawing of a graph $G(V, E)$. The *rotation* $\rho_D(v)$ (or $\rho(v)$ when D is clear from the context) of a vertex v in D is the clockwise cyclic order of edges incident to v, given as a sequence (that is to be interpreted circularly) of the second vertices of all edges at v. (Note that if $G = K_n$ then $\rho(v)$ is a cyclic permutation of $V \setminus \{v\}$). The *rotation system* (abbrev. RS) of D is the set of rotations of all vertices of D and is denoted by $\mathcal{R}(D)$. We consider two rotation systems to be equivalent if one can be obtained from the other by relabeling and optional inversion of all rotations. Further, we call a rotation system *realizable* if it is the rotation system of a good drawing of a complete graph. The following two results imply that for complete graphs, the rotation system uniquely determines the weak isomorphism class of a good drawing (see also [9]), a property that is central to our work.

Theorem 1 (Pach, Tóth [16]) *The rotation system of a good drawing of the complete graph determines the pairs of crossing edges.*

Theorem 2 (Gioan [4]) *The set of crossing pairs of edges determines the equivalence class of the rotation system of a good drawing of the complete graph.*

Note that this is, in general, only true for complete graphs: Determining the crossing number of a (general) graph with a predefined rotation system is NP-complete [19]. A result similar to the above ones is also known for isomorphism classes:

Theorem 3 (Kynčl [9]) *Two good drawings are isomorphic iff there exists a bijection between their vertices such that (i) they are weakly isomorphic, (ii) for each edge, the order of crossings along its image is the same, and (iii) for each crossing the radial order of the edge parts emanating to the four involved vertices is the same (or inverted for all crossings).*

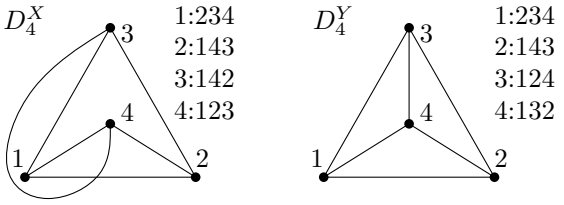


Figure 1: The two different drawings of K_4, with their rotation systems.

K_4 has only two (weak) isomorphism classes, see Figure 1. We denote them by D_4^X and D_4^Y. The basic observation leading to Theorem 1 is that the sub-drawing induced by four points has a rotation system equivalent to $\mathcal{R}(D_4^X)$ if the four points are involved in a crossing, and a rotation system equivalent to $\mathcal{R}(D_4^Y)$ otherwise. (Therefore, Property (iii) in Theorem 3 is also determined by the rotation system for drawings of the complete graph.) The other direction (Theorem 2) is slightly more involved and requires considering also 5-tuples. Unless stated otherwise, we will consider only good drawings of complete graphs (and their rotation systems). We have:

Observation 1 *When given the rotation around three vertices in a drawing of K_4, the relative position of these three vertices in the rotation around a fourth vertex v is determined.*

We generate all rotation systems of size n by *extending* the ones of size $n-1$ in the following way. In the sequence representing the rotation around every vertex, we place the new vertex v_n in all possible ways. Each choice also determines parts of $\rho(v_n)$ by Observation 1. The relative order of two vertices might be different when considering different 4-tuples (which indicates that the choice is invalid) and therefore all 4-tuples containing v_n have to be checked. Hence, we obtain a set of rotation systems where each rotation system restricted to any four vertices is either the one of D_4^X or D_4^Y. We call such a rotation system *consistent*. Still, there exist non-realizable consistent rotation systems. For K_5, there are five (weak) isomorphism classes, and two non-realizable consistent rotation systems. For $n \geq 6$, there are more isomorphism classes than weak isomorphism classes. We describe our approach for checking realizability, which is also used for enumerating all isomorphism classes, in the next section.

To ensure that no two equivalent rotation systems are stored, we guess a vertex that is given the label 1. Then we guess a second vertex to label all vertices from 2 to n, either counterclockwise or clockwise around the first one. This way, we obtain $2n(n-1)$ different labelings. Each labeling gives a matrix consisting of the n rotations. We use the lexicographically smallest one for storing the rotation system. Hence, duplicates can be filtered easily.

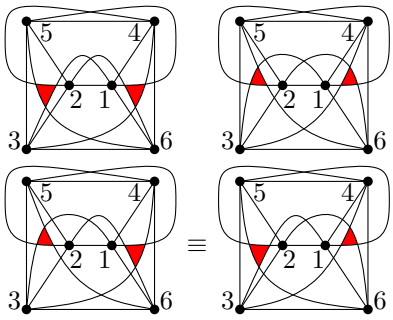


Figure 2: Four drawings of the same rotation system. The two at the bottom are also isomorphic (consider the labeling horizontally mirrored), the others are not.

3 Realizability and Enumeration

It remains to decide realizability of a rotation system and, in case of a positive decision, to count the number of its isomorphism classes. Deciding whether a rotation system of K_n can be realized as a good drawing can be done in polynomial time [10]. Since we also want to enumerate all non-isomorphic drawings of each rotation system, we use a less sophisticated approach that, using properties of the rotation system, works fast for small instances. The basic idea is to use a backtracking algorithm to incrementally build a good drawing, which is represented as a doubly-connected edge list. This algorithm can be used for both checking realizability and for obtaining all realizations of a rotation system.

Similar to recognizing equivalent rotation systems, we use a lexicographically smallest labeling to check isomorphism. However, finding a "fingerprint" for the isomorphism class is more complicated. Consider Theorem 3 (i) and (ii). The first part of the fingerprint is the lexicographically smallest rotation system. The labeling of the vertices defines an order on them, which, in turn, gives a lexicographic order on the edges. For each edge e, from smallest to largest, we list the indices of the edges that cross e, in the order when going from the smaller to the larger vertex. However, there are, in general, several lexicographically smallest labelings for a rotation system, which could give different sequences for the edge crossings. Hence, for a given good drawing, we have to check all such labelings of the rotation system to obtain the lexicographically smallest sequence of edge crossings. An example is given in Figure 2, showing four drawings of one rotation system, of which two are isomorphic.

4 Applications

The numbers of (weak) isomorphism classes are given in Table 1.

n	realizable RS	non-iso. drawings	non-iso. drawings per RS
3	1	1	1 . . . 1
4	2	2	1 . . . 1
5	5	5	1 . . . 1
6	102	121	1 . . . 3
7	11 556	46 999	1 . . . 57
8	5 370 725	502 090 394	1 . . . 46 571
9	7 198 391 729		1 . . . $> 2.3\times10^{10}$

Table 1: The numbers of weak isomorphism classes and non-isomorphic good drawings of K_n, in total and per RS.

4.1 Simple Hamiltonian Cycles

For $n = 7$, Rafla's numbers [20] match ours, confirming the conjecture that every good drawing has a simple (i.e., crossing-free) Hamiltonian cycle for $n \leq 7$. In addition, we verified the conjecture for $n \leq 9$.

4.2 Maximum Number of Crossings

As every drawing of K_4 has at most one crossing, there are at most $\binom{n}{4}$ crossings in a good drawing. But in contrast to complete geometric graphs (where only the set with all points in convex position attains this bound), there exist many weak isomorphism classes with this maximum number of crossings. We call them *max-crossing* drawings. Harborth and Mengersen [8] already considered max-crossing drawings, enumerating all 15 non-isomorphic ones for K_6. Kynčl [9] gives a lower bound of $2^{n-5}\frac{(n-3)!}{n}$ for the number of max-crossing realizable RS, but no upper bounds better than that for all realizable RS are known. Table 2 gives the numbers obtained from our data. Observe that all max-crossing realizable RS can be obtained by extending only max-crossing realizable RS. Therefore, we can go beyond the $n = 9$ barrier by extending only such systems. Note the slight difference to the question in [11, Problem 2], asking for the number of max-crossing consistent RS.

It is known that every good drawing of K_n contains a max-crossing sub-drawing of size $\Omega(\log^{1/8} n)$ [14] (in fact, the bound is given for two particular max-crossing graphs). Table 2 also lists the number of realizable RS with no max-crossing 5-tuple, showing that no such RS of size larger than 12 exists.

4.3 Crossing Number of K_{13}

The crossing number of K_{13} is known to be between 219 [12] and 225 [18]. For odd n, the crossing number has the same parity as $H(n)$ [13]. For a drawing D of K_n with $\mathrm{cr}(D)$ crossings there exists a vertex v s.t. $\mathrm{cr}(D \setminus v) \leq \left\lfloor \frac{n-4}{n}\mathrm{cr}(D) \right\rfloor$ [18]. This allows us to obtain the exact value for $\mathrm{cr}(K_{13})$ by only extending rotation systems with few crossings. By this we were able

n	max-crossing realiz. RS	max-crossing drawings	realiz. RS without 5-crossing 5-tuple
4	1	1	2
5	2	2	3
6	10	15	33
7	115	1 477	606
8	2 657	8 373 474	19 195
9	82 957		449 188
10	3 226 173		4 208 379
11			4 162 266
12			32 290
13			0

Table 2: The number of realizable rotation systems with the maximum number of crossings and the number of sets with no 5-tuple with 5 crossings.

to show that $\mathrm{cr}(K_{13}) \in \{223, 225\}$. For obtaining all rotation systems where K_{13} has at most 223 crossings (if they exist), it is sufficient to extend all rotation systems of K_9 with at most 38 crossings, with intermediate RS for $n = 10, 11, 12$ of at most 64, 102, and 154 crossings. The computations are ongoing.

4.4 Empty Triangles

In a good drawing D, a 3-cycle spanned by three edges of D is called an *empty triangle* if the interior of one of its sides does not contain any vertices of D. Let $\Delta(n)$ be the minimum number of empty triangles over all good drawings of K_n. Harborth [7] showed $2 \leq \Delta(n) \leq 2n - 4$, asking whether this upper bound is tight. The currently best known lower bound of n is given in [3], where also tightness of the upper bound is stated for $n \leq 8$. Using our data, we could extend the positive answer to Harborth's question for $n = 9$.

5 Conclusion

We described the generation of all weakly isomorphic good drawings of K_n for $n \leq 9$. The obtained data allowed us to investigate several open existential and extremal problems for such drawings. We expect the data to be helpful for settling further questions in this area, like the crossing number of K_{13} or the question of which RS maximize the number of non-isomorphic drawings.

Acknowledgments

We thank Jan Kynčl for helpful comments.

References

[1] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar. Shellable drawings and the cylindrical crossing number of K_n. *Discrete Comput. Geom.*, 52(4):743–753, 2014.

[2] O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating order types for small point sets with applications. *Order*, 19(3):265–281, 2002.

[3] O. Aichholzer, T. Hackl, A. Pilz, P. Ramos, V. Sacristán, and B. Vogtenhuber. Empty triangles in good drawings of the complete graph. In *Mexican Conference on Discrete Mathematics and Computational Geometry*, pages 21–29, 2013.

[4] E. Gioan. Complete graph drawings up to triangle mutations. In *Graph-Theoretic Concepts in Computer Science, 31st International Workshop (WG 2005), Revised Selected Papers*, volume 3787 of *LNCS*, pages 139–150. Springer, 2005.

[5] H.-D. O. F. Gronau and H. Harborth. Numbers of nonisomorphic drawings for small graphs. *Congr. Numer.*, 71:105–114, 1990.

[6] J. L. Gross and T. W. Tucker. *Topological Graph Theory*. Wiley, New York, NY, USA, 1987.

[7] H. Harborth. Empty triangles in drawings of the complete graph. *Discrete Math.*, 191:109–111, 1998.

[8] H. Harborth and I. Mengersen. Drawings of the complete graph with maximum number of crossings. *Congr. Numer.*, 88:225–228, 1992.

[9] J. Kynčl. Enumeration of simple complete topological graphs. *Eur. J. Comb.*, 30(7):1676–1685, 2009.

[10] J. Kynčl. Simple realizability of complete abstract topological graphs in P. *Discrete Comput. Geom.*, 45(3):383–399, 2011.

[11] J. Kynčl. Improved enumeration of simple topological graphs. *Discrete Comput. Geom.*, 50(3):727–770, 2013.

[12] D. McQuillan, S. Pan, and R. B. Richter. On the crossing number of K_{13}. *ArXiv e-prints*, July 2013. arxiv:1307.3297.

[13] D. McQuillan and R. B. Richter. A parity theorem for drawings of complete and complete bipartite graphs. *Amer. Math. Monthly*, 117(3):267–273, 2010.

[14] J. Pach, J. Solymosi, and G. Tóth. Unavoidable configurations in complete topological graphs. *Discrete Comput. Geom.*, 30(2):311–320, 2003.

[15] J. Pach and G. Tóth. Which crossing number is it anyway? *J. Combin. Theory Ser. B*, 80(2):225–246, 2000.

[16] J. Pach and G. Tóth. How many ways can one draw a graph? *Combinatorica*, 26(5):559–576, 2006.

[17] J. Pammer. Rotation systems and good drawings. Master's thesis, Graz University of Technology, 2014.

[18] S. Pan and R. B. Richter. The crossing number of K_{11} is 100. *J. Graph Theory*, 56(2):128–134, 2007.

[19] M. J. Pelsmajer, M. Schaefer, and D. Štefankovič. Crossing numbers of graphs with rotation systems. *Algorithmica*, 60(3):679–702, 2011.

[20] N. H. Rafla. *The good drawings D_n of the complete graph K_n*. PhD thesis, McGill University, Montreal, 1988.

A Linear-Time Algorithm for the Queue-Numbers of Proper Triangulated Cacti*

Toru Hasunuma†

Abstract

We present a linear-time algorithm for computing the queue-numbers of proper triangulated cacti.

1 Introduction

A *k-queue layout* of a graph $G = (V, E)$ consists of a vertex-ordering $\sigma : V(G) \mapsto \{1, 2, \ldots, |V(G)|\}$ and an assignment $\rho : E(G) \mapsto \{1, 2, \ldots, k\}$ of the edges to k queues such that no two edges assigned to the same queue nest, i.e., for each $i \in \{1, 2, \ldots, k\}$ and any two edges $\{u, v\}, \{x, y\} \in \rho^{-1}(i)$, it does not hold that $\sigma(u) < \sigma(x) < \sigma(y) < \sigma(v)$. The minimum number of queues for a queue layout of G is the *queue-number* $\mathsf{qn}(G)$ of G. A graph G is called a *k-queue graph* if $\mathsf{qn}(G) \leq k$. Figure 1 illustrates a

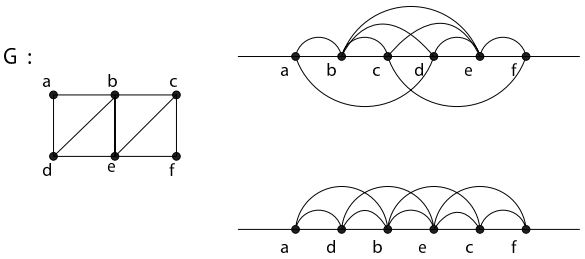


Figure 1: Queue layouts of a graph G.

2-queue layout and a 1-queue layout of a graph G, where the vertices are placed from left to right on a horizontal line according to the vertex-ordering and the edges above (respectively, below) the horizontal line are assigned to the first (respectively, second) queue. The notion of queue layouts was originally introduced by Heath, Leighton, and Rosenberg [10, 11]. Queue layouts of graphs find applications in several areas of computer science such as fault-tolerant computing [14] and three-dimensional graph drawing [3, 6]. Until now, upper bounds on the queue-number have been proved for many graph classes such as complete graphs, complete bipartite graphs, trees [11], outerplanar graphs [10], planar graphs [1, 2], 2-trees [13], graphs with bounded tree-width [3], graphs of bounded genus [4], hypercubes [7], and underlying graphs of iterated line digraphs [8].

*This work was supported by JSPS KAKENHI 25330015.

† Institute of Socio-Arts and Sciences, The University of Tokushima.

The challenging open problem on queue layouts is the conjecture posed by Heath et al. [10, 11] that every planar graph can be laid out using $O(1)$ queues. Di Battista, Frati, and Pach [1] proved that every planar graph can be laid out using $O(\log^4 n)$ queues (and also mentioned that the upper bound can be improved to $O(\log^2 n)$), which improves the previously known bound of $O(\sqrt{n})$ [5]. Dujmović [2] recently proved an $O(\log n)$ upper bound for the queue-number of a planar graph, and moreover Dujmović, Morin and Wood [4] presented poly-logarithmic upper bounds for graphs of bounded genus. The problem of laying out planar graphs using a constant number of queues still remains open, while if we restrict ourselves to outerplanar graphs, such a problem has been solved, i.e., Heath et al. [10] proved that every outerplanar graph can be laid out using two queues. Heath and Rosenberg [11] also characterized 1-queue graphs as arched leveled planar graphs and proved that the problem of recognizing 1-queue graphs is NP-complete. Since the planarity can be efficiently tested [12], the problem of computing the queue-number of a planar graph is NP-hard. In [9], it has been shown that such a problem can be solved in linear-time if we restrict ourselves to maximal outerplanar graphs. A triangulated cactus is a graph in which every block is a maximal outerplanar graph. Thus, the class of triangulated cacti is a subclass of the outerplanar graphs and a superclass of the maximal outerplanar graphs. A triangulated cactus G is called proper if G has no vertex with degree one. In this paper, we extend the result shown in [9] on maximal outerplanar graphs to the class of proper triangulated cacti, i.e., we present a linear-time algorithm for computing the queue-numbers of proper triangulated cacti.

2 Preliminaries

A *maximal outerplanar graph* is an outerplanar graph to which we cannot add a new edge while preserving the outerplanarity. A *leaf* is a vertex with degree one. A *block* of a graph is a maximal connected subgraph without a cut-vertex. A block with at least three vertices is a *cyclic block*, while a block with two vertices is a K_2*-block*. For a connected graph G, the *block-cut-vertex tree* $BCT(G)$ is the graph whose vertices are blocks and cut-vertices of G and in which two vertices are adjacent if and only if one is a cut-vertex and the other is a block containing the cut-vertex. A *leaf-block* of G is a block corresponding to a

leaf of $BCT(G)$. A *triangulated cactus* is a graph in which every block is a maximal outerplanar graph. A triangulated cactus is *proper* if it has no leaf. A *caterpillar* is a tree T such that the graph obtained from T by deleting all the leaves is a path. A path with n vertices is denoted by P_n. For a vertex-ordering σ of G and $u, v \in V(G)$, we may write $u <_\sigma v$ instead of $\sigma(u) < \sigma(v)$. Let σ and σ' be two vertex-orderings of G. If there exists an automorphism ϕ of G such that $\sigma(v) = \sigma'(\phi(v))$ for all $v \in V(G)$, then σ and σ' are *isomorphic* and we write $\sigma \cong \sigma'$. Let H be a subgraph of G. The vertex-ordering of H obtained from the vertex-ordering σ of G is called the *restricted vertex-ordering* of H with respect to σ and is denoted by σ_H.

3 Queue-Numbers of Extended Fan Graphs

We first consider queue-numbers of specific proper triangulated cacti.

Definition 1 *A* fan graph F_n *is a maximal outerplanar graph with n vertices such that there exists a vertex with degree* $n-1$. *The* center vertex *of* F_n *is a vertex with degree* $n-1$ *in* F_n. *An* extended fan graph $F(n_1, n_2, \ldots, n_k)$, *where* $3 \leq n_1 \leq n_2 \leq \cdots \leq n_k$, *is the graph obtained from k fan graphs* $F_{n_1}, F_{n_2}, \ldots, F_{n_k}$ *by identifying the center vertices of* $F_{n_1}, F_{n_2}, \ldots, F_{n_k}$ *with the same vertex.*

The queue-numbers of extended fan graphs can be completely determined as follows. Note that if G contains a subgraph whose queue-number is two, then $\mathsf{qn}(G) = 2$.

Lemma 1

- $\mathsf{qn}(F(7)) = \mathsf{qn}(F(4,5)) = \mathsf{qn}(F(3,4,4)) = 1$.
- $\mathsf{qn}(F(8)) = \mathsf{qn}(F(3,6)) = \mathsf{qn}(F(5,5)) = 2$.
- $\mathsf{qn}(F(3,3,5)) = \mathsf{qn}(F(4,4,4)) = \mathsf{qn}(F(3,3,3,3)) = 2$.

In particular, a vertex-ordering of a 1-queue layout of $F(3,3,3)$ is essentially unique.

Lemma 2 *Let* $V(F(3,3,3)) = \{v_i \mid 0 \leq i \leq 6\}$ *such that* v_0 *is the center vertex and* $\{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6\} \in E(F(3,3,3))$. *Every vertex-ordering for a 1-queue layout of* $F(3,3,3)$ *is isomorphic to the vertex-ordering* σ *defined as* $v_1 <_\sigma v_2 <_\sigma v_3 <_\sigma v_0 <_\sigma v_4 <_\sigma v_5 <_\sigma v_6$.

4 Fundamental Properties of a 1-Queue Layout of a Graph

Lemma 3 *Let G be a 1-queue graph which has a triangle* K_3 *and a path* P_n *as disjoint subgraphs. For any vertex-ordering* σ *of a 1-queue layout of G, it does not hold that* $\sigma^{-1}_{P_n}(1) <_\sigma < \sigma^{-1}_{K_3}(1) <_\sigma \sigma^{-1}_{K_3}(3) <_\sigma \sigma^{-1}_{P_n}(n)$.

From Lemma 3, we can see that in any 1-queue layout of a graph, there is no path through a triangle. Thus, we have the next corollary.

Corollary 4 *Let G be a 1-queue connected graph. Let* σ *be a vertex-ordering of a 1-queue layout of G. If we add a* K_3 *with* $V(K_3) = \{x, y, z\} \cap V(G) = \emptyset$ *so that* $\sigma^{-1}(1) <_\sigma x <_\sigma y <_\sigma z <_\sigma \sigma^{-1}(|V(G)|)$, *then the resultant vertex-ordering does not induce a 1-queue layout of* $G \cup K_3$.

If we restrict ourselves to maximal outerplanar graphs, we can strengthen Corollary 4 as follows.

Lemma 5 *Let G be a 1-queue maximal outerplanar graph. Let* σ *be a vertex-ordering of a 1-queue layout of G. If we add a* K_2 *with* $V(K_2) = \{x, y\} \cap V(G) = \emptyset$ *so that* $\sigma^{-1}(1) <_\sigma x <_\sigma y <_\sigma \sigma^{-1}(|V(G)|)$, *then the resultant vertex-ordering does not induce a 1-queue layout of* $G \cup K_2$.

Besides, we can show the following lemma.

Lemma 6 *Let G be a 1-queue graph which has triangles* K_3 *and* K'_3 *as disjoint subgraphs. For any vertex-ordering* σ *of a 1-queue layout of G such that* $\sigma^{-1}_{K_3}(1) <_\sigma \sigma^{-1}_{K'_3}(1)$, *it holds that* $\sigma^{-1}_{K_3}(2) <_\sigma \sigma^{-1}_{K'_3}(1)$ *and* $\sigma^{-1}_{K_3}(3) <_\sigma \sigma^{-1}_{K'_3}(2)$.

Namely, there are essentially two possible layouts of two disjoint triangles in any 1-queue layout of a graph.

5 Two-Edge-Connected Triangulated Cacti

In this section, we consider the case that a triangulated cactus G has no K_2-block, i.e., G is 2-edge-connected.

Using Lemma 5, we can show the following lemma.

Lemma 7 *Let* G' *be the graph obtained from a 1-queue maximal outerplanar graph G by adding a triangle at a vertex x of G. If* G' *can be laid out using one queue with respect to a vertex-ordering* σ', *then* $\sigma'_G(x) \in \{1, 2, |V(G)| - 1, |V(G)|\}$.

By Lemmas 5, 6, and 7, the next lemma is obtained.

Lemma 8 *Let* G' *be the graph obtained from a maximal outerplanar graph G by adding three triangles at three vertices injectively. Then,* $\mathsf{qn}(G') = 2$.

Thus, any cyclic block in a 1-queue 2-edge-connected triangulated cactus has at most two cut-vertices. Besides, using Lemmas 1 and 2, we can show the following result.

Lemma 9 *Let G be a 1-queue 2-edge-connected triangulated cactus. Then, there are at most three cyclic blocks in G which contain the same cut-vertex. If there are three cyclic blocks in G which contain the same cut-vertex, then one of them is a triangle which contains only one cut-vertex.*

From Lemmas 8 and 9, we can see that for a 1-queue 2-edge-connected triangulated cactus G, $BCT(G)$ is a caterpillar with the maximum degree $\Delta \leq 3$. Applying Lemmas 5, 6, and 7, we have the following lemma.

Lemma 10 *Let B_1, B_2, B_3 be three cyclic blocks of a triangulated cactus G such that B_1 and B_2 (respectively, B_2 and B_3) have the same cut-vertex x (respectively, y), where $x \neq y$. If G can be laid out using one queue with respect to a vertex-ordering σ such that $\sigma(x) < \sigma(y)$, then* $\max_{v \in V(B_1)} \sigma(v) < \min_{v \in V(B_3)} \sigma(v)$.

By Lemma 10, we may assume a direction for layouts of blocks in a 1-queue layout of a 2-edge-connected triangulated cactus G. Let $(B_0, x_1, \ldots, x_p, B_p)$ be a longest path in $BCT(G)$, where B_i is a cyclic block and x_j is a cut-vertex such that $\sum_{1 \leq i \leq p} |V(B_i)|$ is the maximum among longest paths in $BCT(G)$. The set $\mathsf{VO}(B_i)$ of vertex-orderings for 1-queue layouts of B_i can be computed in $O(|V(B_i)|)$ time by using the algorithm in [9]. From results in [9], we can see that $|\mathsf{VO}(B_i)| \leq 3$. Thus, by using Lemma 7, we can check in a constant time whether a vertex-ordering in $\mathsf{VO}(B_i)$ is consistent with a vertex-ordering in $\mathsf{VO}(B_{i+1})$. Therefore, we can compute $\mathsf{qn}(G)$ in $O(|V(G)|)$ time by employing the dynamic programing approach.

6 Proper Triangulated Cacti

Let $G_{1,1,1}$ be the graph obtained from three triangles K_3^1, K_3^2, K_3^3 by adding a new vertex x and joining x to a vertex $v_i \in V(K_3^i)$ for $1 \leq i \leq 3$. For a 3-tuple $(i, j, k) \neq (1, 1, 1)$ of positive integers, let $G_{i,j,k}$ be the graph obtained from $G_{1,1,1}$ by replacing the edge $\{x, v_1\}$ (respectively, $\{x, v_2\}$ and $\{x, v_3\}$) with the path P_i (respectively, P_j and P_k). Applying Lemmas 3 and 6, we have the next fact.

Lemma 11 $\mathsf{qn}(G_{i,j,k}) = 2$.

From Lemma 11 (and Lemma 8), the following lemma is obtained, where an end-cut-vertex is a cut-vertex which has at most one non-leaf-block.

Lemma 12 *Let G be a 1-queue proper triangulated cactus. Then, $BCT(G)$ is a caterpillar with $\Delta \leq 5$ such that except for leaf-blocks adjacent to an end-cut-vertex, every leaf-block is a fan graph whose center vertex is a cut-vertex.*

A *path-component* of a 1-queue proper triangulated cactus G is a maximal connected subgraph in the graph obtained from G by deleting every maximal 2-edge-connected subgraph (2-edge-connected component) except for a cut-vertex incident to a bridge. Let H_i be an extended fan graph for $1 \leq i \leq k$. Let $P(H_1, \ldots, H_k)$ be the graph obtained from the path $P_k = (v_1, \ldots, v_k)$ and H_i $(1 \leq i \leq k)$ by identifying the center vertex of H_i with v_i for $1 \leq i \leq k$. From Corollary 4 and Lemma 6, we may assume a direction for layouts of $H_1, \ldots, H_k$ in a 1-queue layout of $P(H_1, \ldots, H_k)$.

Lemma 13 *For any vertex-ordering σ of a 1-queue layout of $P(H_1, \ldots, H_k)$, if $\sigma^{-1}_{H_{k-1}}(1) <_\sigma \sigma^{-1}_{H_k}(1)$, then for all $1 \leq i < k$, it holds that $\sigma^{-1}_{H_i}(|V(H_i)| - 1) <_\sigma \sigma^{-1}_{H_{i+1}}(1)$ and $\sigma^{-1}_{H_i}(|V(H_i)|) <_\sigma \sigma^{-1}_{H_{i+1}}(2)$.*

Definition 2 *For a vertex-ordering σ in a 1-queue layout of an extended fan graph H, a triangle K_3 with $V(K_3) = \{a, b, c\}$, where a is the center vertex of H, is called* right-toppling *(respectively,* upright *and* left-toppling*) if $\sigma_{K_3}(a) = 1$ (respectively, 2 and 3).*

Definition 3 *Let G be a 1-queue extended fan graph. Then, G is* one-sided toppling *(respectively,* two-sided toppling*) if for any vertex-ordering σ of a 1-queue layout of G, G has an upright triangle and either a right-toppling triangle or a left-toppling triangle (respectively, an upright triangle, a right-toppling triangle, and a left-toppling triangle).*

Except for $F(3) = K_3$ and $F(4)$, the class of 1-queue extended fan graphs is divided into two types.

Lemma 14 *$F(5)$, $F(3,3)$, and $F(3,4)$ are one-sided toppling, while $F(6)$, $F(4,4)$, and $F(3,3,3)$ are two-sided toppling.*

Lemma 15 *In any vertex-ordering σ of a 1-queue layout of $P(H_1, \ldots, H_k)$ with $\sigma^{-1}_{H_{k-1}}(1) <_\sigma \sigma^{-1}_{H_k}(1)$, the following properties hold for $1 \leq i < k$.*

- *If H_{i+1} has an upright triangle, then H_i has no right-toppling triangle.*
- *If H_{i+1} has a left-toppling triangle, then H_i has no upright triangle and no right-toppling triangle.*

Using Lemma 15, we can characterize 1-queue path-components $P(H_1, \ldots, H_k)$ as follows.

Lemma 16 *Let $S = \{H_1, \ldots, H_k\}$ be a set of extended fan graphs. Let N_{one} (respectively, N_{two}) be the number of one-sided toppling (respectively, two-sided toppling) extended fan graphs in S. Then, $\mathsf{qn}(P(H_1, \ldots, H_k)) = 1$ if and only if one of the following conditions holds:*

- *$N_{\text{one}} + N_{\text{two}} \leq 1$.*
- *$N_{\text{one}} = 2$, $N_{\text{two}} = 0$, and for the one-sided toppling extended fan graphs H_p and H_q, where $1 \leq p < q \leq k$, every H_i $(p < i < q)$ is a triangle.*

Besides, we can show the next lemma.

Lemma 17 *Let G_k be the graph obtained from $P(H_1, \ldots, H_p)$ and $P(H'_1, \ldots, H'_q)$ by joining the center vertices of H_p and H'_1 by a path of length k, where $k \geq 2$. Then, $\mathsf{qn}(G_k) = 1$ if and only if $\mathsf{qn}(P(H_1, \ldots, H_p)) = \mathsf{qn}(P(H'_1, \ldots, H'_q)) = 1$.*

By Lemmas 16 and 17, we can determine the queue-number of any path-component of a proper triangulated cactus. In order to check the consistency of queue layouts of a path-component and a 2-edge-connected component, we can use the following lemma.

Lemma 18 *Let G' be the graph obtained from a 1-queue maximal outerplanar graph G by adding the path P_3 at a vertex x of G. If G' can be laid out using one queue with respect to a vertex-ordering σ', then $\sigma'_G(x) \leq 4$ or $|V(G)| - 3 \leq \sigma'_G(x)$.*

7 A Linear-Time Algorithm

Combining the results in Sections 5 and 6, we can design a linear-time algorithm for the queue-numbers of proper triangulated cacti.

Let G be a proper triangulated cactus. We first compute the block-cut-vertex tree $BCT(G)$ of G. If $BCT(G)$ is not a caterpillar with $\Delta \leq 5$, then $\mathsf{qn}(G) := 2$. If $|V(BCT(G))| = 1$, then apply the algorithm [9] for maximal outerplanar graphs. Let $(x_1, B_2, x_2, \ldots, B_{p-1}, x_{p-1})$ be the path obtained from $BCT(G)$ by deleting all the leaves, where B_i $(1 < i < p)$ is a block and x_i $(1 \leq i < p)$ is a cut-vertex. If there exists a leaf-block B' (respectively, B'') containing x_1 (respectively, x_{p-1}) in which there exists a triangle not containing x_1 (respectively, x_{p-1}), then set B_1 as B' (respectively, B_p as B''). Otherwise, set B_1 (respectively, B_p) as a largest block containing x_1 (respectively, x_p). For $(B_1, x_1, B_2, x_2, \ldots, , B_{p-1}, x_{p-1}, B_p)$, we iterate the following manipulations.

Let $(B_{j+1}, x_{j+1}, \ldots, x_{k-1}, B_k)$, where $j < k$, be the path corresponding to a 2-edge-connected component. If there exist B_i $(j + 1 < i < k)$ with $\deg_{BCT(G)}(B_i) \geq 3$, x_i $(j < i < k)$ with $\deg_{BCT(G)}(x_i) \geq 4$, or a leaf-block $B \notin \{B_{j+1}, B_k\}$ with $|V(B)| \geq 4$ which contains x_i $(j < i < k)$, then $\mathsf{qn}(G) := 2$. Compute the set $\mathsf{VO}(B_k)$ of vertex-orderings for 1-queue layouts of B_k that are consistent with a vertex-ordering in $\mathsf{VO}(x_k)$ if $k < p$. If $\mathsf{VO}(B_k) = \emptyset$, then $\mathsf{qn}(G) := 2$. Otherwise, iterate the next manipulations for $j < t < k$ in a dynamic programing approach while $\mathsf{qn}(G) \neq 2$.

1. Compute the set $\mathsf{VO}(B_t)$ of vertex-orderings for 1-queue layouts of B_t that are consistent with a vertex-ordering in $\mathsf{VO}(B_{t+1})$.

2. If $\mathsf{VO}(B_t) = \emptyset$, then $\mathsf{qn}(G) := 2$.

Let $(B_{i+1}, x_{i+1}, \ldots, x_{j-1}, B_j)$, where $i < j$, be the path corresponding to a path-component P_C which precedes $(B_{j+1}, x_{j+1}, \ldots, x_{k-1}, B_k)$. If there exists a leaf-block containing x_ℓ $(i < \ell < j)$ which is not a fan graph whose center vertex is x_ℓ, then $\mathsf{qn}(G) := 2$. Let $G[x_j]$ denote the graph obtained from the subgraph induced by the closed neighborhood of x_j in G by deleting a leaf. Compute the set $\mathsf{VO}(x_j)$ of vertex-orderings for 1-queue layouts of $G[x_j]$ that are consistent with a vertex-ordering in $\mathsf{VO}(B_{j+1})$. By using Lemmas 16 and 17, we can check whether P_C can be laid out using one queue or not. Here, we regard $G[x_j]$ as a two-sided toppling (respectively, one-sided) extended fan graph if for any vertex-ordering in $\mathsf{VO}(x_j)$, $G[x_j]$ has a left-toppling triangle (respectively, $G[x_j]$ has an upright triangle, and there exists a vertex-ordering in $\mathsf{VO}(x_j)$ such that $G[x_j]$ has no left-toppling triangle). This part of checking the queue-number of a path-component can be represented by a finite automaton. If there exists a 1-queue layout of P_C, then we set $\mathsf{VO}(x_i)$ as the set of vertex-orderings for 1-queue layouts of $G[x_i]$ which are consistent with a 1-queue layout of P_C.

Our algorithm can also be applied to a triangulated cactus G such that the graph obtained from G by deleting all the leaves is a proper triangulated cactus.

References

[1] G. Di Battista, F. Frati, and J. Pach. On the queue number of planar graphs. Proceedings of 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2010, pp. 365–374.

[2] V. Dujmović. Graph layouts via layered separators. 2013. arXiv: 1302.0304

[3] V. Dujmović, P. Morin, and D.R. Wood. Layout of graphs with bounded tree-width. SIAM J. Comput. 34 (2005) 553–579.

[4] V. Dujmović, P. Morin, and D.R. Wood. Layered separators for queue layouts, 3D graph drawing and nonrepetitive coloring. Proceedings of 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2013, pp. 280–289.

[5] V. Dujmović and D.R. Wood. On linear layouts of graphs. Discrete Math. & Theoret. Comput. Sci. 6 (2004) 339–358.

[6] E. Di Giacomo, G. Liotta, and H. Meijer. Computing straight-line 3D grid drawings of graphs in linear volume. Computational Geometry 32 (2005) 26–58.

[7] P. Gregor, R. Škrekovski, and V. Vukašinović. On the queue-number of the hypercube. SIAM J. Discrete Math. 26 (2012) 77–88.

[8] T. Hasunuma. Queue layouts of iterated line directed graphs. Discrete Applied Math. 155 (2007) 1141–1154.

[9] T. Hasunuma and A. Haruna. A linear time algorithm for the queue-numbers of maximal outerplanar graphs. Abstracts of 28th European Workshop on Computational Geometry 2012 (EuroCG2012), pp. 37–40.

[10] L.S. Heath, F.T. Leighton, and A.L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. SIAM J. Discrete Math. 5 (1992) 398-412.

[11] L.S. Heath and A.L. Rosenberg. Laying out graphs using queues. SIAM J. Comput. 21 (1992) 927–958.

[12] J.E. Hopcroft and R.E. Tarjan. Efficient planarity testing. J. ACM 21 (1974) 549–568.

[13] S. Rengarajan and C.E.V. Madhavan. Stack and queue number of 2-trees. Proceedings of 1st International Conference on Computing and Combinatorics, LNCS vol. 959, pp. 203–212, 1995, Springer.

[14] A.L. Rosenberg. The Diogenes approach to testable fault-tolerant arrays of processors. IEEE Trans. Comput. C-32 (1983) 902–910.

Simple strategies versus optimal schedules in multi-agent patrolling

Akitoshi Kawamura* Makoto Soejima†

Abstract

Suppose that we want to patrol a fence (line segment) using k mobile agents with given speeds $v_1, \ldots, v_k$ so that every point on the fence is visited by an agent at least once in every unit time period. A simple strategy where the ith agent moves back and forth in a segment of length $v_i/2$ patrols the length $(v_1+\cdots+v_k)/2$, but it has been shown recently that this is not always optimal. Thus a natural question is to determine the smallest c such that a fence of length $c(v_1+\cdots+v_k)/2$ cannot be patrolled. We give an example showing $c \geq 4/3$ (and conjecture that this is the best possible).

We also consider a variant of this problem where we want to patrol a circle and the agents can move only clockwise. We can patrol a circle of perimeter rv_r by a simple strategy where the r fastest agents move at the same speed. We give an example where we can achieve the perimeter of $1.05 \max_r rv_r$ (and conjecture that this constant can be arbitrary big).

We propose another variant where we want to patrol a single point under the constraint that each agent $i = 1, \ldots, k$ can visit the point only at a predefined interval of a_i or longer. This problem can be reduced to the discretized version where the a_i are integers and the goal is to visit the point at every integer time. It is easy to see that this discretized patrolling is impossible if $1/a_1 + \cdots + 1/a_k < 1$, and that there is a simple strategy if $1/a_1 + \cdots + 1/a_k \geq 2$. Thus we are interested in the smallest c such that patrolling is always possible if $1/a_1 + \cdots + 1/a_k \geq c$. We prove that $\alpha \leq c < 1.546$, where $\alpha = 1.264\ldots$ (and conjecture that $c = \alpha$). We also discuss the computational complexity of related problems.

1 Introduction

In *patrolling problems*, a set of mobile agents are deployed in order to protect or supervise a given area, and the goal is to leave no point unattended for a long period of time. Besides being a well-studied task in robotics and distributed algorithms, patrolling raises interesting theoretical questions [4]. Recent studies [2, 6, 3] have shown that finding an optimal strategy is not at all straightforward, even when the terrain to be patrolled is as simple as it could be. We continue this line of research in three basic settings: patrolling a line segment, a circle, and a point. We will be particularly interested in the ratio by which the best schedule could outperform the simple strategy for each problem.

1.1 Fence patrolling

In 2011, Czyzowicz et al. [2] proposed the following problem:

Fence Patrolling Problem. We want to patrol a fence (line segment) using k mobile agents. We are given the speed limits of the agents $v_1, \ldots, v_k$ and the *idle time* $T > 0$. For each point x on the fence and time $t \in \mathbb{R}$, there must be an agent who visits the point x during the interval $[t, t+T)$. How long can the fence be?

Formally, a fence is an interval $[0, L]$, and a *schedule* is a k-tuple $(a_1, \ldots, a_k)$ of functions, where each $a_i \colon \mathbb{R} \to \mathbb{R}$ satisfies $|a_i(s) - a_i(t)| \leq v_i \cdot |s-t|$ for all s, $t \in \mathbb{R}$. It *patrols* the fence with idle time T if for any time $t \in \mathbb{R}$ and any location $x \in [0, L]$, there are an agent i and a time $t' \in [t, t+T)$ such that $a_i(t') = x$.

Note that if we can patrol a fence of length L with idle time T, we can patrol a fence of length αL with idle time αT by scaling, for any $\alpha > 0$. Thus, we are only interested in the ratio of L and T. Unless stated otherwise, we fix the idle time to $T = 1$.

We also note that in previous work [2, 6], a schedule was defined as functions on the halfline $[0, +\infty)$ (instead of $\mathbb{R}$) and the requirement for patrolling was that each location be visited in every length-T time interval contained in this halfline. Our slight deviation from this definition is justified in Section 2 of the full version [7].

Czyzowicz et al. [2] discussed the following simple strategy that patrols a fence of length $(v_1+\cdots+v_k)/2$ (with idle time 1), and proved that no schedule can patrol more than twice as long a fence as this strategy:

Partition-based strategy. Divide the fence into k segments, the ith of which has length $v_i/2$. The agent i moves back and forth in the ith segment.

They conjectured that this gives the optimal schedule. However, Kawamura and Kobayashi [6] exhibited a setting of speed limits $v_1, \ldots, v_k$ and a schedule that patrols a fence slightly longer than the partition-based strategy. Thus, the following natural question arises: what is the biggest ratio between the optimal

*University of Tokyo
†University of Tokyo

schedule and partition-based strategy? Formally, we want to determine the smallest constant c such that no schedule can patrol a fence that is c times as long as the partition-based strategy does.

Czyzowicz et al.'s result [2] says that $1 \leq c \leq 2$, and their conjecture was that $c = 1$. Kawamura and Kobayashi's example shows that $c \geq 42/41$. Later this lower bound was improved to $25/24$ [1, 3]. In Section 2, we will further improve the lower bound to $4/3$. We conjecture that $c = 4/3$.

1.2 Unidirectional circle patrolling

In Section 3, we will discuss another problem proposed by Czyzowicz et al. [2]:

Unidirectional Circle Patrolling Problem. We want to patrol a circle using k mobile agents. We are given the speed limits $v_1, \ldots, v_k$ of the agents. For each point x on the circle and time $t \in \mathbb{R}$, there must be an agent who visits the point x during the interval $[t, t+1)$. Each agent i is allowed to move along the circle in clockwise direction with arbitrary speed between 0 and its speed limit v_i, but it is not allowed to move in the opposite direction. How long can the perimeter of the circle be?

They conjectured that the following strategy is optimal:

Runners strategy. Without loss of generality, we can assume that $v_1 \geq \cdots \geq v_k$. If all the fastest r agents move at constant speed v_r and placed equidistantly, we can patrol a perimeter of length rv_r. By choosing the optimal r, we can achieve the perimeter $\max_r rv_r$.

However, Dumitrescu et al. [3] constructed an example where this strategy is not optimal. We conjecture that this is not even a constant-ratio approximation strategy. Formally, we conjecture that for any constant c, there exist $v_1, \ldots, v_k$ such that we can patrol a perimeter of $c \max_r rv_r$. We will define a problem that is equivalent to this conjecture. Also, we will prove that this is true for $c = 1.05$.

1.3 Point patrolling

In Section 4, we propose a new problem that we call Point Patrolling Problem. In a sense, this is a simplification of the Fence Patrolling Problem. In this problem, agents patrol a single point instead of a fence. In this case, it is natural to set a lower bound on the intervals between two consecutive visits by an agent instead of restricting its speed. Formally, we study the following problem:

Point Patrolling Problem. We want to patrol a point using k mobile agents. We are given the lower bounds $a_1, \ldots, a_k$ on the intervals between two consecutive visits of the agents. A *schedule* is a k-tuple of sets $S_1, \ldots, S_k \subseteq \mathbb{R}$, where S_i means the set of times at which the ith agent visits the point. Thus, if t_1 and t_2 are two distinct elements of S_i, they must satisfy $|t_1 - t_2| \geq a_i$. This schedule *patrols* the point with *idle time* T if for any time $t \in \mathbb{R}$, there are an agent i and a time $t' \in [t, t+T)$ such that $t' \in S_i$. How small can the idle time be?

It turns out that this problem can be reduced to a decision problem that asks whether it is possible to visit the point at each integer time under the constraint that each agent $i = 1, \ldots, k$ can visit the point only at a predefined interval of at least $a_i \in \mathbb{N}$. We will see the relation between the amount $1/a_1 + \cdots + 1/a_k$ and this problem.

In Section 5, we will analyze the complexity of this discretized problem.

2 A schedule patrolling a long fence

The following theorem says that for any $c < 4/3$, there exists a schedule that patrols a fence c times as long as the partition-based strategy. This improves the same claim for $c < 25/24$ established previously [1, 3].

Theorem 1. *For any $c < 4/3$, there are settings of speed limits $v_1, \ldots, v_k$ and a schedule that patrols a fence of length $c(v_1 + \cdots + v_k)/2$ (with idle time 1).*

Proof. We construct, for any positive integers n and L, a schedule that patrols a fence of length L with idle time 1 using $n + L - 1$ agents with speed 1 and nL agents with speed $1/(2n-1)$. Note that with the partition-based strategy, the same set of agents would patrol (with idle time 1) a fence of length $\frac{1}{2}(n + L - 1 + nL/(2n-1))$. The ratio between L and this approaches $4/3$ when $1 \ll n \ll L$, and hence we have the theorem. The schedule that proves our claim is as follows (Figure 1):

- Each of the $n + L - 1$ agents A_i $(-n < i < L)$ with speed 1 visits the locations i and $i + n - 1/2$ alternately (at its maximal speed); it is at location i at time 0. (This means that some agents occasionally step out of the fence $[0, L]$; to avoid this, we could simply modify the schedule so that they stay at the end of the fence for a while.)

- Each of the nL agents $B_{i,j}$ $(0 \leq i < L, 0 \leq j < n)$ with speed $1/(2n-1)$ visits the locations $i + 1/2$ and $i + 1$ alternately (at its maximal speed); it is at location $i + 1/2$ at time $j + 1/2$.

It can be verified that this schedule patrols the fence (see the full version [7]). □

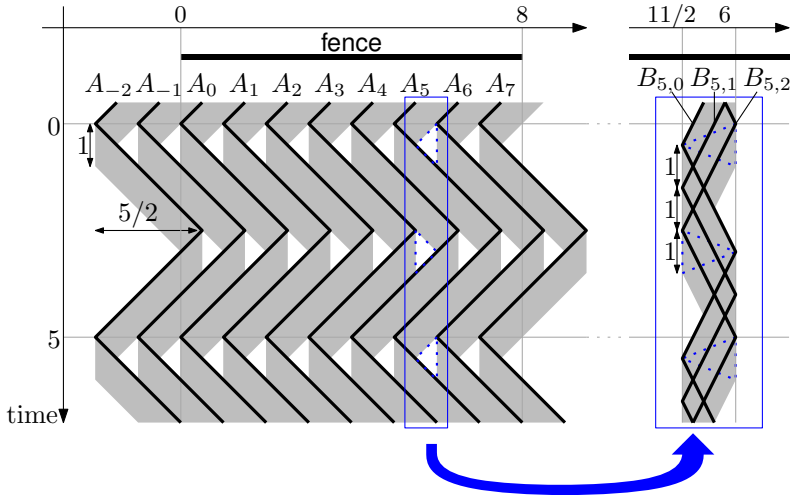


Figure 1: The strategy in the proof of Theorem 1 when $n = 3$ and $L = 8$. The trajectories of the agents are the thick solid lines, and the regions they cover (the points that have been visited during the past unit time, see appendix) are shown shaded. The $n + L - 1$ faster agents $A_{-n+1}, \ldots, A_{L-1}$ (left) move back and forth with period $2n - 1$, but leave some triangular regions (dotted) uncovered. These regions are covered by the nL slow agents $B_{0,0}, \ldots, B_{n-1,L-1}$ (right; scaled up horizontally for clarity).

We conjecture that this constant 4/3 is the best possible. That is,

Conjecture 2. *No schedule can patrol a fence that is more than* 4/3 *times as long as the partition-based strategy.*

3 Circle patrolling

We start by defining (c, k)-*sequences*, whose existence is closely related to the Circle Patrolling Problem as we will show in Lemma 3.

For a real number $c > 1$ and a positive integer k, a (c, k)-sequence is a k-tuple of sets $S_1, \ldots, S_k \subseteq \mathbb{R}$ with $S_1 \cup \cdots \cup S_k = \mathbb{R}$ such that for each i,

1. the set S_i is a union of non-overlapping intervals $S_i = \bigcup_{j \in \mathbb{Z}} [a_{i,j}, b_{i,j}]$;
2. the length of each interval in S_i is at most $1/(ci - 1)$, i.e., $b_{i,j} - a_{i,j} \leq 1/(ci - 1)$;
3. the distance between two consecutive intervals in S_i is *exactly* 1, i.e., $a_{i,j+1} - b_{i,j} = 1$.

See the full version [7] for a proof of the following lemma.

Lemma 3. *Let* $c > 1$.

1. *If* k *agents with speed limits* $1, 1/2, \ldots, 1/k$ *can patrol a circle of perimeter* c*, then there is a* (c, k)*-sequence.*
2. *If there is a* (c, k)*-sequence, then* k *agents with speed limits* $1, 1/2, \ldots, 1/k$ *can patrol a circle of perimeter* $c/2$.

In particular, the runners strategy for circle patrolling is not a constant-ratio approximation strategy if and only if for any constant c, there exists k such that a (c, k)-sequence exists.

Using a computer program, we have found a $(2.1, 122)$ (see the full version [7] for details). Thus,

Theorem 4. *There exist* $v_1, \ldots, v_k$ *and a schedule that patrols a circle with perimeter* $1.05 \max_r rv_r$.

We conjecture that for any constant c, there exist an integer k and a (c, k)-sequence. Equivalently,

Conjecture 5. *The runners strategy does not have a constant approximation ratio. Formally, for any constant* c*, there exist* $v_1, \ldots, v_k$ *and a schedule that patrols a circle with perimeter* $c \max_r rv_r$.

4 Point patrolling

In this section, we will discuss Point Patrolling Problem. First, we observe that this problem can be reduced to a problem in which time is also discrete. Consider a decision version of this problem. That is, you are given T, and you need to decide whether the idle time can be at most T. We can reduce the original problem to this decision problem by binary search. This decision problem can be discretized in the following way:

Discretized Point Patrolling Problem. There are k agents and they want to patrol a point. We are given positive integers $a_1, \ldots, a_k$. The interval between two consecutive visits by the ith agent must be at least a_i. A schedule is called *good* if at each integer time

the point is visited by at least one agent. Determine whether there exists a good schedule.

For simplicity, we call $(a_1, \ldots, a_k)$ *good* if there exists a good strategy in Discretized Point Patrolling Problem, and otherwise call it *bad*. It is not hard to see the following.

Theorem 6. *Agents with intervals $(a_1, \ldots, a_k)$ can achieve the idle time of T for the (non-discretized) Point Patrolling Problem if and only if $(\lceil a_1/T \rceil, \ldots, \lceil a_k/T \rceil)$ is good.*

For the rest of this section, we will be interested in sufficient conditions for $(a_1, \ldots, a_k)$ to be good or bad.

Theorem 7. *If $\sum_{i=1}^{k} 1/a_i < 1$, $(a_1, \ldots, a_k)$ is bad.*

Proof. Let M be a sufficiently big integer. Out of any consecutive M integer times, the ith agent can visit the point at most $\lceil M/a_i \rceil$ times. If $(a_1, \ldots, a_k)$ is good, the sum of $\lceil M/a_i \rceil$ must be at least M, but this contradicts $\sum_{i=1}^{k} 1/a_i < 1$ when M is sufficiently big. □

On the other hand, the following gives a sufficient condition for $(a_1, \ldots, a_k)$ to be good when $a_1, \ldots, a_k$ are powers of 2:

Lemma 8. *If $\sum_{i=1}^{k} 1/2^{b_i} \geq 1$, $(2^{b_1}, \ldots, 2^{b_k})$ is good.*

Proof. We prove the lemma by induction on k. Since $\sum_{i=1}^{k} 1/2^{b_i} \geq 1$, at least one of the following conditions hold:

- For some i, $b_i = 0$. In this case, $(2^{b_1}, \ldots, 2^{b_k})$ is obviously good.
- There exist distinct i, j such that $b_i = b_j = t$. Let S be a set of integers. If an agent with interval $2d$ can visit the point at all elements in S, there exists a schedule of two agents with intervals d such that for each element in S, at least one agent visits the point. Thus, we can replace two agents with intervals 2^t with an agent with interval 2^{t-1}. This replacement doesn't change the inverse sum of intervals, and by the assumption of the induction $(2^{b_1}, \ldots, 2^{b_k})$ is good. □

These two lemmas give a polynomial-time 2-approximation algorithm for the (non-discretized) Point Patrolling Problem.

In the rest of this section, we focus on the relation between Discretized Point Patrolling Problem and the amount $\sum_{i=1}^{k} 1/a_i$.

Theorem 9. *If $\sum_{i=1}^{k} 1/a_i \geq 2$, $(a_1, \ldots, a_k)$ is good.*

Proof. Let b_i be an integer that satisfies $a_i \leq 2^{b_i} < 2a_i$. Since $\sum_{i=1}^{k} \frac{1}{2^{b_i}} \geq \sum_{i=1}^{k} \frac{1}{2a_i} \geq 1$, by lemma 8, $(2^{b_1}, \ldots, 2^{b_k})$ is good. Therefore, $(a_1, \ldots, a_k)$ is also good. □

This constant 2 can be improved, as shown the following theorem (see the full version [7] for a proof).

Theorem 10. *If $\sum_{i=1}^{k} 1/a_i > 1.546$, $(a_1, \ldots, a_k)$ is good.*

On the other hand, the constant cannot be smaller than $\sum_{i=0}^{\infty} 1/(2^i + 1) = 1.264\ldots$ (again, see the full version [7] for a proof):

Theorem 11. *$(2, 3, 5, \ldots, 2^k + 1)$ is bad.*

We suspect that this cannot be improved:

Conjecture 12. *Let $\alpha := \sum_{0}^{\infty} 1/(2^i + 1) \approx 1.264$. If $\sum_{i=1}^{k} 1/a_i > \alpha$, $(a_1, \ldots, a_k)$ is good.*

5 Complexity of problems related to point patrolling

We have discussed approximation algorithms for patrolling problems. This is because patrolling problems look unsolvable in polynomial time. In this section, we will try to justify this intuition. Ideally, we should prove NP hardness of patrolling problems, but we failed to prove that. Instead, we prove NP-completeness of some problems related to Discretized Point Patrolling Problem in the full version [7].

References

[1] K. Chen, A. Dumitrescu, and A. Ghosh. On fence patrolling by mobile agents. In *Proc. 25th Canadian Conference on Computational Geometry* (CCCG), 2013.

[2] J. Czyzowicz, L. Gąsieniec, A. Kosowski, and E. Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In *Proc. 19th European Symposium on Algorithms* (ESA), 2011, LNCS 6942, pp. 701–712.

[3] A. Dumitrescu, A. Ghosh, and C. D. Tóth. On fence patrolling by mobile agents. *Electronic Journal of Combinatorics*, 21, P3.4, 2014.

[4] A. Dumitrescu and C. D. Tóth. Computational Geometry Column 59. *ACM SIGACT News*, 45(2), 2014.

[5] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[6] A. Kawamura and Y. Kobayashi. Fence patrolling by mobile agents with distinct speeds. *Distributed Computing*, to appear.

[7] A. Kawamura and M. Soejima. Simple strategies versus optimal schedules in multi-agent patrolling. To appear in *Proc. 9th International Conference on Algorithms and Complexity* (CIAC), 2015. Also available at arXiv:1411.6853, 2014.

[8] Štefan Znám. On exactly covering systems of arithmetic sequences. *Mathematische Annalen*, 180(3), 227–232, 1969.

[9] Z. Sun. On disjoint residue classes. *Discrete Mathematics*, 104(3), 321–326, 1992.

Continuous Geometric Algorithms for Robot Swarms with Multiple Leaders

Maximilian Ernestus* Sándor Fekete* Michael Hemmer* Dominik Krupke*

Abstract

We consider the problem of building a dynamic and robust network between mobile terminals with the help of a large swarm of robots in the continuous Euclidean plane. Individually, the robots have limited capabilities, both in terms of global information and computation. We propose a set of local *continuous* algorithms that together produce a generalization of a Euclidean Steiner tree. At any stage, the resulting overall shape achieves a good compromise between local thickness, global connectivity, and flexibility to further continuous motion of the terminals.

1 Introduction

Robot navigation is one of the classical application areas for computational geometry. How can we gather the geometric information that is necessary for orienting ourselves in a known or unknown environment? How can we carry out geometric computations efficiently, and how can we optimize specific objectives? Without a doubt, the close interaction between theory and practice for these challenges has motivated major progress, both in robotics and in computational geometry. Even without a specific focus on robotics, a relatively new area of algorithmics has arisen from considering not just a single active agent, but a whole group or even swarm. Swarm robotics combines classical robotics with distributed algorithms and many aspects of wireless sensor networks.

Traditionally, computational geometry has focused on discrete algorithms. In this paper, we demonstrate that a more continuous (not event-based) approach is able to lead to interesting and non-trivial geometric algorithms. In particular, we consider a large swarm of mobile robots with very simple individual capabilities. Motion is continuous, as is interaction and response between different robots.

The challenge is to combine two fundamentally opposite objectives: How can we develop local self-stabilizing mechanisms that allow the swarm to stay locally well connected, even when being pulled apart by several distant and mobile sites?

*Department of Computer Science, TU Braunschweig, Germany. maximilian@ernestus.de, s.fekete@tu-bs.de, mh-saar@gmail.com, d.krupke@tu-bs.de

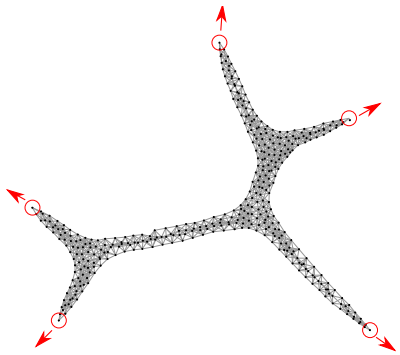


Figure 1: A robust robot swarm emulating a Steiner tree between five diverging leader robots.

Related Work. Even in a centralized and static setting with full information, we have to deal with the well-known NP-hard problem of finding a good Steiner tree [3]. There is a large body of work on geometric swarm behavior; for lack of space, we only mention Chazelle [1] for flocking behavior, and Fekete et al. [2, 5] for geometric algorithms for static sensor networks. As far as we know, only Hamann and Wörn [4] have explicitly considered the construction of Steiner Trees by a robot swarm. For static terminals, they start with an exploratory network; as soon as all terminals are connected, only best paths are kept and locally optimized. More specific references are given in Section 3.1, where they are used as building blocks.

This Paper. We propose a set of local, self-stabilizing algorithms that maintain a dynamic and robust network between leader robots. The algorithms ensure that the swarm adopts the directions of multiple leaders, while preserving a uniform thickness along the edges of the Steiner tree. We demonstrate the usefulness of this approach by simulations with a swarm of 400 robots, five leaders and various failure rates.

2 Preliminaries

For a finite set of robots $\mathcal{R}$ with an externally controlled subset of leader robots $\mathcal{L} \subsetneq \mathcal{R}, |\mathcal{L}| \ll |\mathcal{R}|$, we want the remaining robots $\mathcal{R} \setminus \mathcal{L}$ to maintain a dynamic and robust network that keeps the swarm connected, even in the presence of random robot failures and arbitrary leader movements. Thus, the over-

all shape of the swarm should form a "thick" Steiner tree among the leaders with the robots $\mathcal{R} \setminus \mathcal{L}$ evenly distributed along the edges, as shown in Figure 1.

Robots have the shape of circles; two of them are connected when within a maximum distance and with an unobstructed line of sight. Robots know the relative positions and orientations of their neighbors and can communicate asynchronously. Each robot has a unique ID; leader IDs are known by all others. Robot's translations and rotations are limited in velocity and acceleration. Communication is possible by broadcasting to immediate neighbors.

The perception of all robots is local; however, due to the known position and orientation difference, each robot can transform vectors of its neighbors to its own coordinate system. We avoid multi-hop transformations to keep errors small.

3 Algorithm

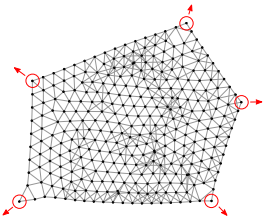
The proposed approach consists of a set of local self-stabilizing mechanisms that either detect a condition or induce a force. The weighted sum of the induced forces determines the robot motion; input for the local mechanisms of the local state and environment of the robot, output is a value for current robot motion. In principle, these mechanisms are continuous. (Our implementation described later updates at 60 Hz.)

We first discuss the base behavior of the robots in Section 3.1, inducing an almost convex swarm shape. This is subsequently improved by leader forces, stability improvement and thickness contraction.

3.1 Base Behavior

Our base behavior consists of three components that result in a swarm shape of a droplet. (i) The *flocking algorithm* of Olfati-Saber [8] considers regular distribution and movement consensus. The algorithm is a stateless equation based on potential fields and is proven to converge. It uses three rules: Attraction to neighbors, repulsion from too close neighbors, and adaption to the velocity of neighbors. We slightly modified the algorithm for better response to additional forces. (ii) An extended version of the *boundary detection* algorithm of McLurkin and Demaine [7], which determines if a robot lies on the boundary and also identifies small holes[1] by using the average angle. (iii) The *boundary tension* of Lee and McLurkin [6], which straightens and minimizes the boundary of the swarm. This is done by simply pushing boundary robots to the middle of its two boundary neighbors.

[1]The method theoretically allows the robots to distinguish exterior and interior boundaries and determine their size, but the limited precision and the convergence time limit this usage.



However, the base behavior without any other forces results in at most convex shapes before losing connectivity. The figure to the right depicts a situation in which the swarm is just about to lose connectivity. For stronger control and more variable shapes, leader forces are introduced.

3.2 Leader Forces

A single leader constitutes the simplest form of swarm control. In this case the swarm motion is determined by the leader's velocity. With multiple (possibly antagonistic) leaders, the swarm is not just steered, but may be stretched to the limit until connectivity is lost. Therefore, each robot needs to find an appropriate balance between the influence of different leaders. See top of Figure 2 for an illustration.

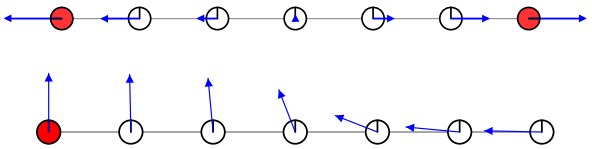


Figure 2: (Top) A one-dimensional scenario with two leaders (red) moving in opposite directions. (Bottom) With increasing distance to the leader, the effect shifts from velocity matching to leader pursuit.

There are two ways of following a leader: either by matching its velocity or by moving towards it. Velocity matching preserves the overall shape of the swarm, but fails with multiple leaders. In addition, there are accumulated losses in accuracy with each hop because the velocity information needs to be passed between robots with noisy sensors. On the other hand, moving towards the leader causes a deformation of the swarm and can also be used to control its shape when multiple leaders are used. However, regions close to the leaders suffer from "compression". We therefore combine both methods by a smooth transition between velocity matching close to the leaders and leader pursuit when further away; see bottom of Figure 2.

In order to achieve the combination of movement *with* the leader and *towards* the leader, three public variables are used for each leader. The **leader distance** is the minimum hop count to the leader. Let $\text{pred}(r)$ be the predecessor in a minimum-hop tree to the leader, which can be the leader itself. The **leader velocity** is the one of $\text{pred}(r)$ for a non-leader, and the robot's own velocity for the leader. The **leader direction** is a normalized direction vector calculated incrementally from the direction to $\text{pred}(r)$ as follows: Each robot takes the *leader direction* of its $\text{pred}(r)$ and merges it with the normalized direction

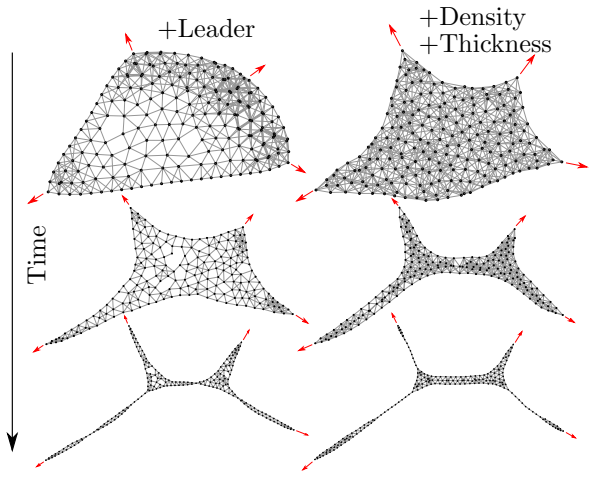


Figure 3: (Left) The basic swarm with leader forces added. (Right) Swarm with stability improvement. Lower swarms are scaled down for better visibility.

to pred(r). If pred(r) is the leader, only the normalized direction to it is used. For computing the leader force, the *leader direction* is scaled to the length of the *leader velocity* and then combined with a *leader distance*-sensitive weighting.

For $\ell \in \mathcal{L}$, let $c_\ell : \mathcal{R} \to \mathbb{R}^2$ be the force on a specific robot and let $d_\ell : \mathcal{R} \to \mathbb{N}$ be its distance to ℓ. The leader forces on robot r are combined as follows:

$$\sum_{\ell \in \mathcal{L}} c_\ell(r) \frac{d_\ell(r)^{-1}}{\sum_{\ell' \in \mathcal{L}} d_{\ell'}(r)^{-1}}.$$

Additionally we provide leaders with too few neighbors with an attraction force, so they do not lose connection to the swarm. This attraction spreads over some distance, but decreases exponentially.

3.3 Stability Improvement

Near Steiner points, connections along concave swarm boundaries may be stretched by boundary forces. When the involved edges approach the upper bound for communication, connections may be disrupted, to the point where the swarm loses connectivity. By adding a thickness-dependent compression force, we reduce neighbor distances without influencing the Steiner-tree shape of the swarm; in effect, this works similar to compression stockings. In the following, we give a heuristic for thickness computation and compression. In order to let the flocking algorithm handle this compression without destroying the regular distribution, we sketch a density distribution heuristic later in this Section. A comparison of a swarm with and without the stability improvement can be seen in Figure 3.

Thickness Contraction. We define the local thickness at a robot as the radius of the largest hop circle

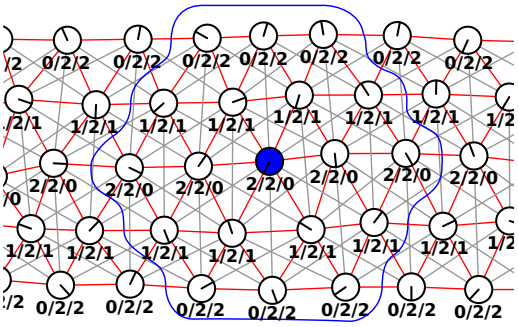


Figure 4: Thickness determination ($b(r)/t(r)/h(r)$) for a limb part. The red edges fulfill the Gabriel graph condition. A largest *hop circle* is marked in blue.

containing it. A hop circle of radius h with robot c as circle center is the set of all robots with a hop count $\leq h$ to c; only robots with distance equal to h may be on the boundary. An example is highlighted in blue in Figure 4.

The relationship between geometric thickness and boundary hop distance may be distorted by long connections that skip over robots. This can be avoided by only considering edges that fulfill the edge condition of the Gabriel graph, meaning that no robot is allowed to be closer to the midpoint of an edge than the robots connected by it. We denote the corresponding reduced neighborhood of a robot r as N'_r.

The following method is a simplified implementation of the thickness metric above, which performed well enough in simulation. It gets by with only three public variables; all circles with its center within a larger circle are ignored.

For this heuristic evaluation of the thickness $t(r)$ of a robot r, we need the hop distance $b(r)$ from the boundary and the circle center distance $h(r)$. Computing the hop distance to the boundary for each robot can easily be achieved by setting $b(r)$ to 0 for all robots on the boundary, while all others take the minimum of their neighbors plus one, as follows

$$b(r) = \begin{cases} 0 & r \text{ on boundary} \\ \min\{b(n)+1 \mid n \in N'_r\} & \text{else} \end{cases}$$

Small holes, that occur frequently but also vanish quickly, are excluded from the boundary, otherwise the value can become too instable. The thickness $t(r)$ is determined as the maximum $b(r)$ within some range $h(r)$, as follows.

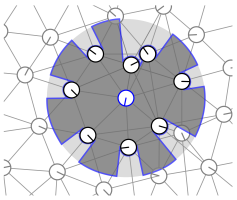
$$t(r) := \max\{\{b(r)\} \cup \{t(n) \mid n \in N'_r \wedge t(n) + \lambda \geq h(n)\}\},$$

where $\lambda \in \mathbb{N}$ is a small constant (e.g. $\lambda = 2$) that tackles the problem of irregular boundaries. If r is a circle center ($t(r) = b(r)$), then the circle center distance $h(r)$ is 0. Otherwise,

$$h(r) := \min\{h(n)+1 \mid n \in N'_r \wedge t(n) = t(r)\}$$

Based on this thickness $t(r)$, the described compression force grows linearly with this $t(r)$. It acts only on robots of large boundaries, so that small holes are not prevented from closing.

Density. The local density of a robot refers to the number of neighbors in relation to its observable area. By introducing an attraction to low and repulsion from high local density neighbors, the overall swarm density is maintained at a specific homogeneous level. It is determined by dividing the number of neighbors by the roughly calculated observable area as depicted in the figure to the right. In order to avoid lumps, robots in collision range are weighted higher. For robots on the boundary the computation is a bit more involved. However, further details are omitted due to limited space.



Let $\rho(r')$ be the averaged local density of robot r', ϱ the optimal density, and N_r the neighbors of r, then the density distribution force for a robot r is given by

$$\sum_{n \in N_r} \overline{p}_r(n) * \phi(\rho(n) - \varrho) \text{ with } \phi(x) = x^3/|x|,$$

where $\overline{p}_r : \mathcal{R} \to \mathbb{R}^2$ is the direction from robot r to a neighbor with the length of the distance for $\rho(n) \leq \varrho$ and of range minus distance else. We do not apply this force to robots on the boundary.

4 Experiments

We validated our approach by conducting experiments with a set of five leaders stretching out a swarm of 400 robots until it disconnects. The performance is measured by the length of the minimal Steiner tree on disconnection (calculated by the Geosteiner software [9]), divided by the theoretically maximal possible length estimated by $|\mathcal{R}'| * \text{range}$, where $\mathcal{R}'$ are the robots that did not fail yet. This would correspond to an optimal but extremely fragile Steiner tree in which *any* node failure disconnects the swarm. Thus, the best possible value of 1 is completely elusive.

For comparison we tested three configurations: Base—only the base behavior as discussed in Section 3.1; Lead—the basic behavior enriched by leader forces as discussed in Section 3.2; All—the final configuration that also incorporates Density and Thickness Contraction as presented in Section 3.3.

The experiments were carried out with 60 iterations per simulated second, a robot diameter of 10 cm and a range of 1.2 m. The maximal robot velocity was set to $1\,\mathrm{m\,s^{-1}}$, but the leaders only moved by at most $0.25\,\mathrm{m\,s^{-1}}$ in order to give the swarm robots the opportunity to react. These parameters are chosen arbitrary but are oriented to the R-One swarm robots of the Rice University.

Failure rate	Base	Lead	All
0	.07 .08 .09	.25 .30 .34	.28 .32 .35
$5 \cdot 10^{-6}$	.07 .08 .09	.25 .28 .32	.26 .29 .33
10^{-5}	.06 .08 .09	.23 .28 .31	.26 .30 .33
$2 \cdot 10^{-5}$	.07 .08 .09	.22 .25 .29	.26 .30 .33

Table 1: Relative Steiner tree sizes reached by first, second, and third quartiles. The failure rate is the probability of each robot to die in each step of the simulation.

For each configuration there were 100 random trials on four different failure rates. The results in Table 1 show that leader forces already produce decent swarm behavior, with survivability four times higher than for the base forces. Without robot losses, it reaches around 30% of the length of the hypothetical optimum. With robot failures, the performance gets weaker with increasing failure probability. The variant with additional stability improvement is slightly better without failures, but is clearly more robust against robot losses.

References

[1] B. Chazelle. The convergence of bird flocking. *J. ACM*, 61(4):21, 2014.

[2] S. P. Fekete and A. Kröller. Geometry-based reasoning for a large sensor network. In *Proc. SoCG*, pages 475–476, 2006.

[3] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing steiner minimal trees. *SIAM J. Appl. Math.*, 32(4):835–859, 1977.

[4] H. Hamann and H. Wörn. Aggregating robots compute: An adaptive heuristic for the euclidean steiner tree problem. In *From Animals to Animats 10*, pages 447–456. Springer, 2008.

[5] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proc. SODA*, pages 1000–1009, 2006.

[6] S. K. Lee and J. McLurkin. Distributed cohesive configuration control for swarm robots with boundary information and network sensing. In *Proc. IROS*, pages 1161–1167. IEEE, 2014.

[7] J. McLurkin and E. D. Demaine. A distributed boundary detection algorithm for multi-robot systems. *Proc. IROS*, pages 4791–4798, 2009.

[8] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. *IEEE Trans. Automat. Contr.*, 51:401–420, 2006.

[9] D. Warme, P. Winter, and M. Zachariasen. Geosteiner 3.1. *Department of Computer Science, University of Copenhagen (DIKU)*, 2001.

Randomized Strategy for Walking in Streets for a Simple Robot

Azadeh Tabatabaei* Mohammad Ghodsi†

Abstract

We consider the problem of walking in an unknown street, for a robot that has a minimal sensing capability. The robot is equipped with a sensor that only detects the discontinuities in depth information (gaps) and can locate the target point as enters in its visibility region. We propose a randomized search strategy that generates a search path for the simple robot to reach the target t, starting from s. Even with such a limited capability, we prove that the expected distance traveled by the robot is at most a constant times longer than the shortest path to reach the target.

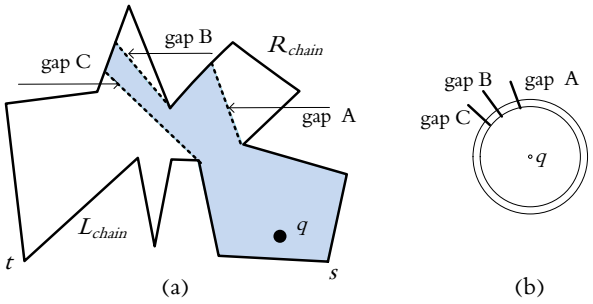


Figure 1: (a) A street polygon. The colored region is the visibility polygon of the point robot q in the street. (b) The position of discontinuities in the depth information (gaps) reported by the sensor.

1 Introduction

Path planning is a basic problem to almost all scopes of computer science; such as computational geometry, online algorithms, robotics and artificial intelligence [10]. Especially, path planning in an unknown environment for which there is no geometric map of the scene is interesting in many real life cases. Robot sensors is the only tool for gathering information in an unknown street. Amount of the information achieved from the environment depends on the capability of the robot. Due to the importance of using simple robot, including low cost, less sensitive to failure, robust against sensing errors and noise, many types of path planning for simple robot have been studied [1, 3, 5].

In this paper, we consider the problem of walking a simple robot in an unknown street. A simple polygon P with two separated vertices s and t is called a street if the left boundary chain L_{chain} and the right boundary chain R_{chain} constructed on the polygon from s to t are mutually weakly visible. In other words each point on the left chain can see at least one point on the right chain and vice versa [8], see Figure 1.

A point robot which its sensor has a minimal capability that can only detect discontinuities in depth information (gaps) and the target point t, starts searching the street. The robot can locate the target as soon as it enters in its visibility region. Also, the robot cannot measure any angles or distances or infer its position, see Figure 1. The goal is to reach the target t using the information gathered through its sensor, starting from s such that the traveled path by the robot is as short as possible.

In order to evaluate the efficiency of a search strategy for the robot, we use the notation of the competitive analysis. The competitive analysis for a strategy that leads the robot is the ratio of the distance traversed by the robot over the shortest distance from s to t, in the worst case. In previous research, Tabatabaei and Ghodsi gave a deterministic algorithm for the simple robot to reach the target t in the street, starting from s with the competitive ratio of 11. Also they showed that 9 is a lower bound for the competitive ratio of each deterministic algorithm [13, 14]

In this paper, we present a randomized strategy for the simple robot to walk in the street. We show that the worst case ratio of the expected distances traveled by the robot to the shortest path form s to t is 6.59 which is almost twice as good as the competitive ratio of the deterministic algorithm.

Related Works: Klein proposed the first competitive algorithm for walking in streets problem for a robot that was equipped with a 360 degrees vision system [8]. Also, Icking et al. presented an optimal search strategy for the problem with the competitive factor of $\sqrt{2}$ [6].

The limited sensing model (gap sensor) that our robot is equipped with, in this research, was first introduced by Tovar et al. [16]. They offered Gap Navigation Tree (GNT) to maintain and update the gaps seen along a navigating path. Other researcher presented some strategies, using GNT, for exploring unknown environments [4, 9, 11]. An optimal search

*Department of Computer Engineering, Sharif University of Technology.

†Sharif University of Technology and School of Computer Science, Institute for Research in Fundamental Sciences (IPM).

strategy with minimum number of turns, for the simple robot equipped with the gap sensor, is presented in [15].

Another minimal sensing model was presented by Suri et al. [12]. They assumed that the simple robot can only sense the combinatorial (non-metric) properties of the environment. The robot can locate the vertices of the polygon in its visibility region, and can report if there is a polygonal edge between them. Despite of the minimal ability, they showed that the robot can accomplish many non-trivial tasks. Then, Disser et al. empowered the robot with a compass to solve the mapping problem in polygons with holes [2].

2 The Sensing Model and Known Properties

At the start point, the point robot reports a cyclically ordered of discontinuities in the depth information (gaps) in its visibility region. Each gap has a label of L or R which displays the direction of the part of the scene that is hidden behind the gap, see Figure 2. Also the robot carries a pebble to mark some location of the street. The robot can orient its heading to each gap and moves towards the gap in an arbitrary number of steps. Also the robot can move towards the pebble and the target as they enter in its visibility region. While the robot moves, combinatorial changes occur in the visibility region of the robot that they are called critical events. There are four types of critical events: appearances, disappearances, merges, and splits of gaps. Appearance and disappearance events occur when the robot crosses inflection rays. An appeared gap, during the movement, corresponds to a portion of the environment that was already visible, but now is not visible. such the gaps are called primitive gaps and the other gaps are non-primitive gaps. Merge and split events occur when the robot crosses bitangent complements, as illustrated in Figure 2.

Now, we express some primary properties of locations of gaps in the walking in streets problem, mostly from [14]. At each point of the search path, if the target is not visible, the robot reports a set of left and right gaps (l-gap and r-gap for abbreviation). Let g_l be the most advanced non-primitive left gap (l-gap) and g_r be the most advanced non-primitive right gap (r-gap), see Figure 3. It is shown in [6, 14], while the target is not visible then it is hidden behind one of the two most advanced gaps. So, if there exist only one of the two gaps (g_r and g_l) then the goal is hidden behind of the gap. Thus, there is no ambiguity and the robot moves towards the gap, see Figure 3(a). When both of g_r and g_l exist, a funnel case arises. At each funnel case there are two convex chains in front of the robot: the left convex chain that lies on L_{chain} of the street, and the right convex chain that lies on R_{chain}, see Figure 3(b). Following property is an important guideline for achieving the target.

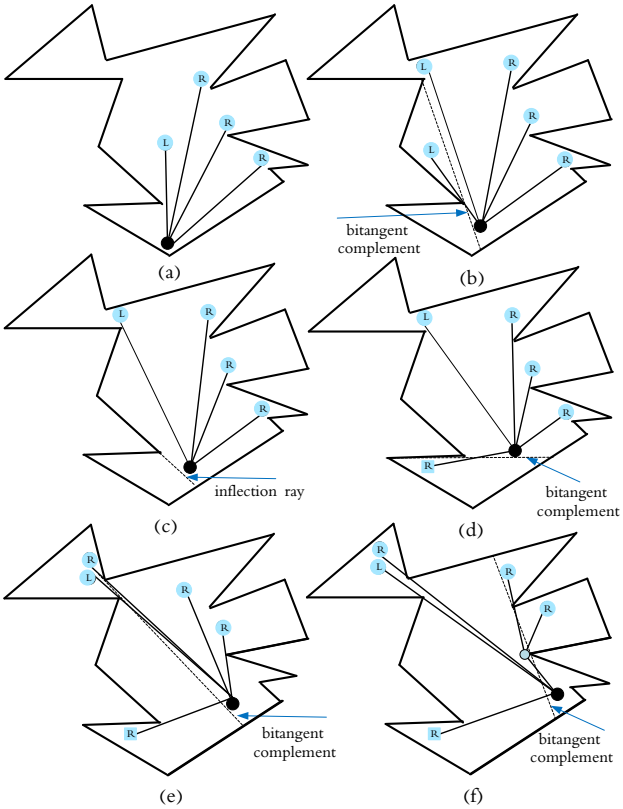


Figure 2: The dynamically changes of the gaps as the robot walks towards a gap. The dark circle is the location of the robot, and squares and other circles denote primitive and non-primitive gaps respectively. (a) Existing gaps at the start point. (b) A split event. (c) A disappearance event. (d) An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

Theorem 1 *Shortest path from s to t lies completely on the left convex chain, or on the right convex chain of the funnel, at each funnel case.*

At each funnel case, the chain which contains the shortest path from s to t is called the exact chain of the funnel.

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

Obviously, if the robot moves towards the left/right most advanced gap, it traces the left/right convex chain.

The following events update the location of g_l and g_r as well as a funnel situation.

1. When the robot crosses a bitangent complement of g_r/g_l and another r-gap/l-gap, then g_r/g_l

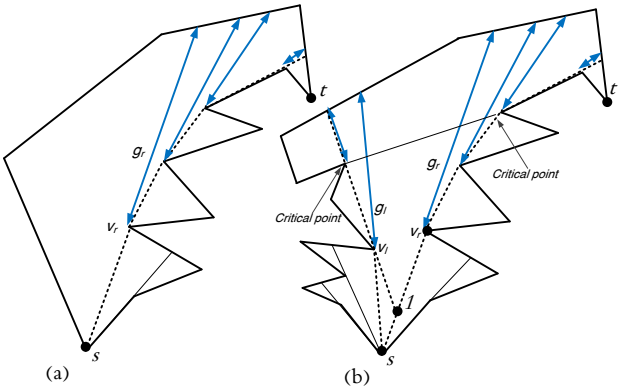


Figure 3: g_r and g_l are the most advanced gaps at the start point s. v_r and v_l are the corresponding reflex vertices. (a) There is only one most advanced gap, at start point s. (b) Sequences of the most advanced gaps may occur, as the robot moves. The funnel situation which ends as soon as the robot reaches each of the critical points. Dotted chains, starting from s, are the two convex chains of the funnel.

splits and will be replaced by the r-gap/l-gap, (point 1 in Figure 3(b)).

2. When the robot crosses a bitangent complement of g_r/ g_l and an l-gap/r-gap, then g_r/g_l splits into two gaps. g_l/g_r will be replaced by the l-gap/r-gap, (point 1 in Figure 4). This point is a critical point in which a new funnel situation starts.

3. When the robot crosses over an inflection ray, each of g_l or g_r which is adjacent to the ray, disappears and is eliminated, (point 2 in Figure 4). This point is a critical point in which a funnel situation may end.

The critical points of chains have the following property.

Lemma 2 *[14] The exact chain of the funnel can be specified as soon as the robot reaches a critical point of the funnel.*

3 Algorithm

Now, we present the randomized strategy for searching the street, from s to t. Since the target is constantly behind one of the most advanced gaps, during the searching, the location of them is maintained and dynamically updated. There are two cases at the start point:

- If only one of the two gaps (g_r and g_l) exists, then the goal is hidden behind the gap. Thus, there is no ambiguity and the robot moves towards the gap until the target is achieved or a funnel situation arises, see Figure 3(a).

- If there is a funnel case, the robot puts a pebble to mark this point as origin. Since it is not sure that the target is behind which of g_r or g_l, it chooses a uniform random variable from $\{0, 1\}$. If the random variable is 1(0), at each stage $i \in \{0, 1, 2, ...\}$, the robot moves 2^i steps along the right (left) convex chain and returns to the origin, then moves 2^{i+1} steps along the left (right) convex chain. The robot repeats moving back and forth along the two convex chain until a critical point of the funnel is achieved. From Lemma 2, the robot can distinguish the exact chain. So, after achievement of the critical point, the robot comes back to the origin and picks the pebble up, and moves along the exact convex chain, see Figure 4. The robot continues to move along the exact chain until the target is achieved or a new funnel situation arises. Note that in a funnel case when the robot moves along a convex chain, the dynamically changes of g_l and g_r are maintained, as explained in the previous section. Furthermore, it has to maintain the comeback path (R_g) to the origin. This path is constructed as follows:

  – When the robot crosses over an inflection ray, a gap appears. If this gap hides the pebble, we refer to this gap as comeback gap (R_g), see Figure 4.

  – When R_g merges with another gap, the comeback gap (R_g) will be a child of the gap.

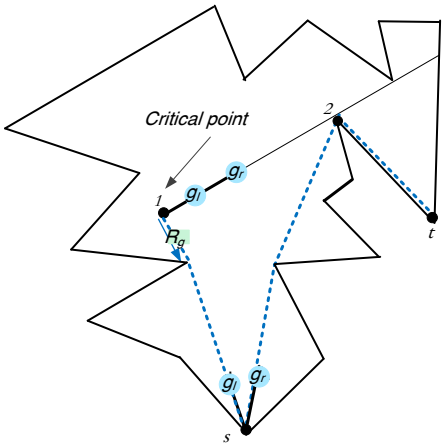


Figure 4: The dotted path is the robot search path. Point 1 is a critical point in which the robot distinguish the exact chain. R_g maintains the comeback path.

3.1 Correctness and Analysis

Now, we show that our strategy generates a search path for the robot to reach the target t. Also, we show that the ratio of expected distance traversed by the robot to the shortest path distance from s to t is a constant number.

When the robot reports only one most advanced gaps, the robot moves towards the gap. So, its path coincides with the shortest path. When a funnel case arises, assume the two convex chains in front of the robot as a line. By our randomized strategy, the robot achieves the critical point of the exact chain. A similar argument, that proves the competitive ratio of the smart cow algorithm [7], can be used to show that expected distance traveled by the robot is 4.59 times as long as the shortest path distance to the point. After achievement of the critical point the robot returns to the origin to pick up the pebble and moves along the exact chain. Then the expected distances traversed by the robot to reach the critical point and picking up the pebble is at most 6.59 times as long as the shortest path.

The following theorem that is an immediate consequence of the above discussion is the main result of this paper.

Theorem 3 *The randomized strategy generates a search path to achieve target t in the street, starting from s, with a competitive ratio of 6.59.*

4 Conclusion

In this paper we proposed a randomized search strategy for walking in streets problem for a point robot that has a minimal sensing capability. The robot can only detect the gaps in the environment and the target. Also the robot has access to a pebble as a marker. Our randomized search strategy generates a search path for the robot with a competitive ratio of 6.59. This competitive ratio is almost twice as good as the competitive ratio of the previous deterministic algorithm.

References

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

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

[3] Fekete, Sandor P., Joseph SB Mitchell, and Christiane Schmidt. *Minimum Covering with Travel Cost.* Journal of Combinatorial Optimization, 24: 32-51, 2003.

[4] Guilamo, L., Tovar, B., LaValle, S. M. *Pursuit-evasion in an unknown environment using gap navigation trees.* In Intelligent Robot's and Systems Proceedings Vol. 4, (pp 3456–3462), 2004.

[5] Hammar, Mikael, Bengt J. Nilsson, and Mia Persson. *Competitive exploration of rectilinear polygons.* Theoretical computer science 354(3): 367–378, 2006.

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

[7] Kao, Ming-Yang, John H. Reif, and Stephen R. Tate. *Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem.* Information and Computation 131(1): 63–79,1996.

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

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

[10] Mitchell, Joseph SB. *Geometric shortest paths and network optimization, Handbook of Computational Geometry.* Elsevier Science Publishers, 1998.

[11] Sachs, S., LaValle, S. M., Rajko, S. *Visibility-based pursuit-evasion in an unknown planar environment.* The International Journal of Robotics Research 23(1): 3–26, 2004.

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

[13] Tabatabaei, Azadeh, and Mohammad Ghodsi. *Walking in Streets with Minimal Sensing.* Journal of Combinatorial Optimization http://link.springer.com/article/10.1007, 2014

[14] Tabatabaei, Azadeh, and Mohammad Ghodsi. *Walking in Streets with Minimal Sensing.* In Combinatorial Optimization and Applications. Springer International Publishing, (pp. 361–372), 2013.

[15] Tabatabaei, Azadeh, and Mohammad Ghodsi. *Optimal Strategy for Walking in Streets with Minimum Number of Turns for a Simple Robot.* Combinatorial Optimization and Applications. Springer International Publishing, (pp. 101–112), 2014.

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

Orienting triangulations*

Boris Albar† Daniel Gonçalves‡ Kolja Knauer§

Abstract

We prove that any triangulation of a surface different from the sphere and the projective plane admits an orientation of its 1-skeleton without sinks such that every vertex has outdegree divisible by three. This confirms a conjecture of Barát and Thomassen (J. of Graph Theory (2006)) and is a step towards a generalization of Schnyder woods to higher genus surfaces.

1 Introduction

The notation and results we use for graphs and surfaces can be found in [9]. We start with some basic definitions:

A *map* (or 2-*cell embedding*) of a multigraph into a surface, is an embedding such that deleting the graph from the surface leaves a collection of open disks, called the *faces* of the map. A *triangulation* is a map of a simple graph (i.e. without loops or multiple edges) where every face is triangular (i.e. incident to three edges). A fundamental result in the topology of surfaces is that every surface admits a map. The *(orientable) genus* of a map on an orientable surface is $\frac{1}{2}(2-n+m-f)$ and the *(non-orientable) genus* of a map on a non-orientable surface is $2-n+m-f$, where n, m, f denote the number of vertices, edges, and faces of the map, respectively. The *Euler genus* k of a map is $2-n+m-f$, i.e., the non-orientable genus or twice the orientable genus. All the maps on a fixed surface have the same genus, which justifies to define the *(Euler) genus of a surface* as the (Euler) genus of any of the maps it admits. In [2] Barát and Thomassen conjectured the following:

Conjecture 1 *Let T be a triangulation of a surface of Euler genus $k \geq 2$. Then T has an orientation of its edges such that each outdegree is at least* 3*, and divisible by* 3*.*

One easily computes that the number of edges m of a triangulation T of a surface of Euler genus k is $3n-6+3k$. So while triangulations of Euler genus less than 2 simply have too few edges to satisfy the conjecture, in [2] the conjecture is proved for the case $k = 2$, i.e., the torus and the Klein bottle. Moreover, they show that any triangulation T of a surface has an orientation such that each outdegree is divisible by 3, i.e, in order to prove the full conjecture they miss the property that there are no sinks.

Barát and Thomassen's conjecture was originally motivated in the context of claw-decompositions of graphs, since given an orientation with the claimed properties the outgoing edges of each vertex can be divided into claws (i.e. copies of $K_{1,3}$), such that every vertex is the center of at least one claw.

Another motivation for this conjecture is, that it can be seen as a step towards the generalization of planar Schnyder woods to higher genus surfaces. A *Schnyder wood* [10] of a planar triangulation is an orientation and a $\{0,1,2\}$-coloring of the *inner* edges satisfying the following *local rule* on every *inner* vertex v: going counterclockwise around v one successively crosses an outgoing 0-arc, possibly some incoming 2-arcs, an outgoing 1-arc, possibly some incoming 0-arcs, an outgoing 2-arc, and possibly some incoming 1-arcs until coming back to the outgoing 0-arc.

Schnyder woods are one of the main tools in the area of planar graph representations and Graph Drawing. They provide a machinery to construct space-efficient straight-line drawings [11, 6], representations by touching T shapes [5], they yield a characterization of planar graphs via the dimension of their vertex-edge incidence poset [10, 6], and are used to encode triangulations efficiently [3]. In particular, the local rule implies that every Schnyder wood gives an orientation of the inner edges such that every inner vertex has outdegree 3 and the outer vertices are sources with respect to inner edges. Indeed, this is a one-to-one correspondance between Schnyder woods and orientations of this kind.

When generalizing Schnyder woods to higher genus surfaces one has to choose which of the properties of planar Schnyder woods are desired to be carried over to the more general situation. Examples are: the efficient encoding of triangulations on arbitrary surfaces [4] and the relation to orthogonal surfaces and small grid drawings for toroidal triangulations [8], which lead to different definitions of generalized Schnyder woods. In [8], the generalized Schnyder woods indeed satisfy the local rule with respect to *all* edges and vertices of a toroidal triangulation and henceforth lead to orientations having outdegree 3 at every vertex. An interesting open problem is to

*This work was supported by the project EGOS, ANR-12-JS02-002-01

†LIRMM, CNRS & Université Montpellier 2
‡LIRMM, CNRS & Université Montpellier 2
§LIF, Université Aix-Marseille

generalize the local rule to triangulations with higher Euler genus in such a way that for some vertices the sequence mentioned in the local rule occurs several times around the vertex. Here, the mere existence of such objects is an open question. Clearly, such a generalized Schnyder wood would yield an orientation as claimed by the conjecture. Thus, proving the conjecture of Barát and Thomassen is a first step into that direction. More towards generalizations of Schnyder woods can be found in [7]. A full version of the present paper is on the arXiv [1].

2 Preliminaries

A map M on a surface $\mathbb{S}$ is characterized by a triple $(V(M), E(M), F(M))$, formed by the vertex, edge and face sets of M. In the following we will restrict to triangulations $T = (V(T), E(T), F(T))$, i.e. the pair $(V(T), E(T))$ is a simple embedded graph such that every face is incident to exactly three edges.

A *submap* M' of T, is a triplet (V', E', F') where $V' \subseteq V(T)$, $E' \subseteq E(T)$ and $F' \subseteq F(T)$. Note that a submap is not necessarily a map. A submap $M' = (V', E', F')$ is *closed* if $uv \in E'$ implies $\{u, v\} \subseteq V'$ and if $f \in F'$ implies $e \in E'$ for any edge e incident to f. The *closure* $cl(M')$ of a submap M' (of T) is the smallest closed submap of T containing M'. The *boundary* $\partial M'$ of a submap M' is the set of edges in $cl(M')$ that are incident to at most one face in M'.

In a submap M' of T a *(boundary) angle* $\widehat{e_0 v e_t}$ at vertex v is an alternating sequence $(e_0, f_1, e_1, \ldots, f_t, e_t)$, for some $t \geq 1$, of edges and faces incident to v (in T) and such that:

- the faces f_i are mutually different, for $1 \leq i \leq t$,
- each face f_i, for $1 \leq i \leq t$, is incident to edges e_{i-1} and e_i,
- both edges e_0 and e_t belong to $cl(M')$,
- but none of the remaining edges, e_i for $0 < i < t$, belong to $cl(M')$, nor any faces f_i, for $0 < i \leq t$.

In the following, a *disk* is a submap M' of T if it is homeomorphic to an (open or closed) topological disk.

Given a triangulation T and a set of vertices $X \subseteq V(T)$, the *induced submap* $T[X]$ is the submap with vertex set X, edge set $\{uv \in E(T) \mid u \in X \text{ and } v \in X\}$, and face set $\{uvw \in F(T) \mid u \in X, v \in X, \text{ and } w \in X\}$.

Given an induced submap $M' = T[X]$ of a triangulation T, and any occurrence of an edge ab in $\partial M'$ (corresponding to angles $\hat{a}$ and $\hat{b}$) there exists a unique vertex c such that there is a face abc in $T \setminus M'$ that belongs to both angles $\hat{a}$ and $\hat{b}$. For any such vertex c (and $ab \in \partial M'$) we define the operation of *stacking* c on M', as adding c to X, i.e., going from $M' = T[X]$ to $M'' = T[X + c]$. In such stacking, let $M' \cap cl(M'' \setminus M')$ be the *neighborhood* of c in M'. As T is simple, note that this neighborhood is either a cycle or a union of paths, one of which with at least one edge (the edge allowing the stacking), and let us respectively call them the *neighboring cycle* and the *neighboring paths* of c in M' (see Figure 1).

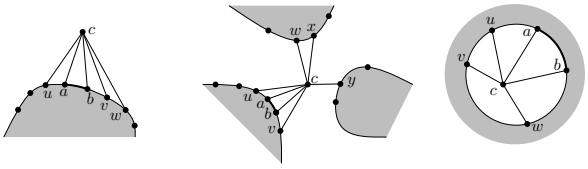


Figure 1: Different scenarios of stacking c to M'. Left: one neighboring path $P_1 = (u, a, b, v, w)$. Middle: three neighboring paths $P_1 = (u, a, b, v)$, $P_2 = (w, x)$, $P_3 = (y)$. Right: A boundary cycle $C = (u, v, w, b, a)$.

3 Proof of Conjecture 1

Let us consider for contradiction a minimal counterexample T. We first provide an outline of the proof.

3.1 Outline

We first prove that one can partition the edges of the triangulation T into the following graphs:

- The *initial graph* I, which is an induced submap containing a non-contractible cycle. Furthermore, I contains an edge $e^* = \{u, v\}$ such that the map $I \setminus e^*$ is a disk D whose underlying graph is a maximal outerplanar graph with only two degree two vertices, u and v (see Figure 2).

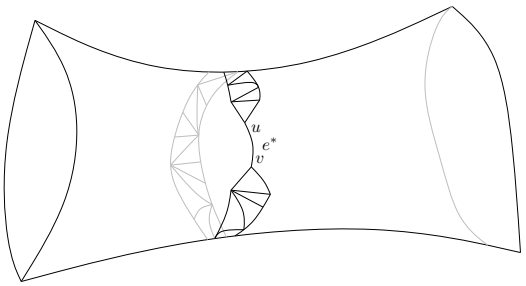


Figure 2: Example of a submap I.

- The *correction graph* B (with blue edges in the figures), which is oriented acyclically in such a way that each vertex of $V(T) \setminus V(I)$ has outdegree 2 in B, while the other vertices have outdegree 0,
- The *last correction path* G (with green edges in the figures), which is a $\{u, v\}$-path.

- The *non-zero graph* R (with red edges in the figures), which is oriented in such a way that all vertices in $(V(T) \setminus V(G)) \cup \{u, v\}$ have out-degree at least 1 in R.

Due to space restrictions, the existence of the initial graph I will not be proven here, we refer to [1] for a proof of this fact. In Section 3.2 we sketch the proof of the existence of the graphs B, G and R (with the mentioned orientations).

Finally, the edges of I, B and G are (re)oriented, to obtain the desired orientation. The orientation of edges in R does not change, as they ensure that many vertices (all vertices of T except the interior vertices of the path G) have non-zero outdegree. The $\{u, v\}$-path G is either oriented from u to v or from v to u, but this will be decided later. However in both cases its interior vertices are ensured to have non-zero outdegree. Hence all vertices are ensured to have non-zero outdegree and it remains to prove that they have outdegree divisible by 3.

We start in Section 3.3 by reorienting the B-arcs in order to ensure that vertices of $V(T) \setminus V(I)$ have outdegree divisible by 3 (this is the same approach as the one used in [2]). In the last step, in Section 3.4, we choose the orientation of the $\{u, v\}$-path G, and we orient I in order to achieve the desired orientation.

3.2 Existence of B, G, and R

We start from I and we incrementally conquer the whole triangulation T by stacking the vertices one by one (this is inspired by [4]). At each step, we will assign the newly explored edges to B, G or R, and we will orient those assigned to B or R. At each step the *explored part* is a submap of T induced by some vertex set X. The explored part is hence the submap denoted $T[X]$ with boundary $\partial T[X]$. The *unexplored part* is the submap $T \setminus T[X]$, and it may consist of several components.

At a given step of this exploration, the graph G may not be an $\{u, v\}$-path yet. In such a case, the graph G will consist of two separate paths G_u and G_v, respectively going from u to u', and from v to v', for some vertices u' and v' on $\partial T[X]$. Here the vertices u' and v' may respectively coincide with vertices u and v, if G_u or G_v is a trivial path on just one vertex.

During the exploration we maintain the following invariants:

(I) The graphs I, B, G, and R partition the edges of $T[X]$.

(II) All interior vertices of $T[X]$ have at least one outgoing R-arc, or two incident G-edges.

(III) The graph B is acyclically oriented in such a way that the vertices of I have outdegree 0, while the other vertices of $T[X]$ have outdegree 2.

Furthermore, to help us in properly finishing the construction of the graphs B, G and R in the further steps, we introduce the notion of *requests* on the angles of $\partial T[X]$. Informally, a G-request (resp. an R-request) for an angle $\hat{a}$ means that in a further step an edge inside this angle will be added in G (resp. in R and oriented from a to the other end). Every angle has at most one request, and an angle having no request is called *free*.

(IV) Every vertex of $(\partial T[X] \setminus \{u', v'\}) \cup \{u, v\}$ having (still) no outgoing R-arc, has an incident angle with an R-request.

(V) If G is not a $\{u, v\}$-path (yet), u' and v', have one incident angle each, say $\hat{u}'$ and $\hat{v}'$, that are consecutive on $\partial T[X]$, and that have a G-request.

(VI) If there is an unexplored disk D', i.e. a component of the unexplored part that is a disk, then there are at least three free angles (of $\partial T[X]$) around D'.

This exploration starts with $T[X] = I$. In this case as all the edges of $T[X]$ are in I and as there are no interior vertices yet, (I), (II) and (III) are trivially satisfied. Since the Euler genus of T is at least 2 there is no unexplored disk, hence (VI) is satisfied. Since $e^* = uv$ appears twice in $\partial T[X]$, the vertices u, v appear twice consecutively in $\partial T[X]$. To achieve (V) and (IV), we assign requests to the vertices of I as in Figure 3.

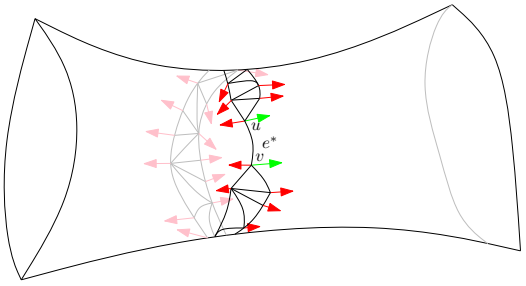


Figure 3: Assigning requests to I in order to satisfy the invariants.

For the rest of the construction in each step we enlarge the explored map $T[X]$ by stacking a vertex x carefully choosen to an edge $e \in \partial T[X]$. We then partition the edges incidents to x into the different graphs B, G and R and update the requests of the vertices adjacent to x in a way to satisfy the different invariants and the previous requests of the vertices. We show that this is always possible by studying all the possible configurations (see Figure 4 for an example where x has only one neighboring path). All the others cases are omitted due to space restrictions.

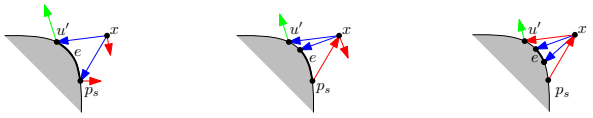


Figure 4: Case where there is only one G-request (on $\hat{u}'$) and where $\hat{p_s}$ has an R-request. The 3 subcases from left to right correspond to the cases where there is 0, 1 and 2 free angles in the neighboring path respectively.

3.3 Reorienting B

Given a partial orientation O of T we define the *demand* of a vertex v as $\text{dem}_O(v) := -\delta^+_{|O}(v) \mod 3$, where $\delta^+_{|O}(v)$ denotes the outdegree of v with respect to O. We want to find an orientation of T with all demands 0.

Recall we will not modify the orientation on R, which guarantees that all vertices in $(V(T) \setminus V(G)) \cup \{u, v\}$ have non-zero outdegree. Furthermore, as G will be oriented either entirely forward or backwards, all its interior vertices will have non-zero outdegree. Hence every vertex of $T[X]$ has non-zero outdegree. Suppose that G is entirely oriented forward.

Now we linearly order vertices in $V(T) \setminus V(I) = (v_1, \ldots, v_\ell)$ such that with respect to B every vertex has its two outgoing B-neighbors among its predecessors and I. Denote by B_i the subgraph of B induced by the arcs leaving $v_i, \ldots, v_\ell$ (before the reorienting). We process $V(T) \setminus V(I)$ from the last to the first element. At a given vertex v_i we look at $\text{dem}_{G \cup R \cup B_i}(v_i)$ and reorient the two originally outgoing B-arcs of v_i in such a way that afterwards $\text{dem}_{G \cup R \cup B_i}(v_i) = 0$ (i.e. $\delta^+_{|G \cup R \cup B_i}(v_i) \equiv 0 \mod 3$). As these B-arcs were heading at I or at a predecessor, the demand on the vertices v_j, with $j > i$, is not modified and hence remains 0.

3.4 Orienting G and I

Denote by O the partial orientation of T obtained after 3.3. Pick an orientation of G (either all forward or all backward) and of $e^* = uv$ such that for the resulting partial orientation O' we have $\text{dem}_{O'}(v) \equiv 1 \mod 3$.

Now, take the triangle Δ of I containing v. Since $D = I \setminus e^*$ is a maximal outerplanar graph with only two degree two vertices, D can be peeled by removing degree two vertices until reaching Δ. When a vertex x is removed orient its two incident edges so that $\text{dem}_{O'}(x) = 0$ (as for B-arcs). We obtain a partial orientation O'', such that all vertices except the ones of Δ have non-zero outdegree divisible by 3.

Since the number of edges of T, and the number of edges of Δ are divisible by 3, the number of edges of $T \setminus \Delta$ is divisible by 3. As this number equals the sum of the outdegrees in O'', and as every vertex out of Δ has outdegree divisible by 3, then the outdegree of Δ's vertices sum up to a multiple of 3. Hence their demands sum up to 0, 3 or 6. As $\text{dem}_{O''}(v) = \text{dem}_{O'}(v) = 1$, the demands of the other two vertices of Δ are either both 1, or 0 and 2. It is easy to see that in either case Δ can be oriented to satisfy all three demands.

References

[1] B. Albar, D. Gonçalves, and K. Knauer, *Orienting triangulations*, arXiv 1412.4979, (2014).

[2] J. Barát and C. Thomassen, *Claw-decompositions and Tutte-orientations*, J. Graph Theory, 52 (2006), pp. 135–146.

[3] O. Bernardi and N. Bonichon, *Intervals in Catalan lattices and realizers of triangulations*, J. Combin. Theory Ser. A, 116 (2009), pp. 55–75.

[4] L. Castelli-Aleardi, E. Fusy, and T. Lewiner, *Schnyder woods for higher genus triangulated surfaces, with applications to encoding*, Discrete & Computational Geometry, 42 (2009), pp. 489–516.

[5] H. de Fraysseix, P. O. de Mendez, and P. Rosenstiehl, *On triangle contact graphs*, Combin. Probab. Comput., 3 (1994), pp. 233–246.

[6] S. Felsner, *Convex drawings of planar graphs and the order dimension of 3-polytopes*, Order, 18 (2001), pp. 19–37.

[7] D. Gonçalves, K. Knauer, and B. Lévêque, *Structure of Schnyder labelings on orientable surfaces*. (in preparation).

[8] D. Gonçalves and B. Lévêque, *Toroidal maps: Schnyder woods, orthogonal surfaces and straight-line representations*, Discrete & Computational Geometry, 51 (2014), pp. 67–131.

[9] B. Mohar and C. Thomassen, *Graphs on surfaces*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 2001.

[10] W. Schnyder, *Planar graphs and poset dimension*, Order, 5 (1989), pp. 323–343.

[11] W. Schnyder, *Embedding planar graphs on the grid*, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '90, 1990, pp. 138–148.

Lattice 3-polytopes with six lattice points*

Mónica Blanco Francisco Santos

Abstract

We completely enumerate lattice 3-polytopes of width larger than one and with exactly 6 lattice points: There are 74 of width 2, two of width 3, and none of larger width.

According to the number of interior points these 76 polytopes divide into 23 tetrahedra with two interior points, 49 polytopes with one interior point and only 4 *hollow* polytopes (two tetrahedra, one quadrangular pyramid and one triangular bipyramid).

1 Introduction

A *lattice polytope* is the convex hull of a finite set of points in $\mathbb{Z}^d$ (or in a d-dimensional lattice). A polytope is *d-dimensional* if it contains $d+1$ affinely independent points. We call *size* of P its number $\#(P \cap \mathbb{Z}^d)$ of lattice points and *volume* of P its volume normalized to the lattice (that is, $d+1$ points form a simplex of volume one if and only if they are an affine lattice basis). More formally, let $p_i \in \mathbb{Z}^d$:

$$\mathrm{vol}(\mathrm{conv}\{p_1,\dots,p_{d+1}\}) := \left|\det \begin{pmatrix} 1 & \dots & 1 \\ p_1 & \dots & p_{d+1} \end{pmatrix}\right|$$

The *width* of a lattice polytope is the minimum of $\max_{x\in P} f(x) - \min_{x\in P} f(x)$, over all possible (non-constant) choices of a linear functional $f : \mathbb{R}^d \to \mathbb{R}$ with $f(\mathbb{Z}^d) \subset \mathbb{Z}$. In other words, the width of P is the minimum lattice distance between two parallel lattice hyperplanes that enclose P. In particular, P has *width one* if its vertices lie in two parallel and consecutive lattice hyperplanes.

Two lattice polytopes P and Q are said *$\mathbb{Z}$-equivalent* or *unimodularly equivalent* if there is an affine unimodular transformation $t : \mathbb{R}^d \to \mathbb{R}^d$ that preserves the lattice and with $t(P) = Q$. We call such a transformation a *$\mathbb{Z}$-equivalence*. Volume, width, and size are obviously invariant modulo $\mathbb{Z}$-equivalence.

In dimension 2, once we fix an $n \in \mathbb{N}$ there are finitely many $\mathbb{Z}$-equivalence classes of lattice 2-polytopes of size n. In dimension 3, in contrast, there are infinitely many classes for each size $n \geq 4$. Still, combining previous results it is easy to show that:

Theorem 1 ([2]) *For each $n \geq 4$, there are finitely many lattice 3-polytopes of width greater than one and size n.*

So, it makes sense to classify separately, for each n, the 3-polytopes of width one and those of larger width. Those of width one are infinitely many, but easy to describe. Of larger width there is none up to $n = 4$ ([9]) and there are exactly 9 for $n = 5$, all of width two ([2]). Here we completely classify 3-polytopes of size $n = 6$ and width > 1, showing that there are exactly 74 of width two, two of width three, and none of larger width (see precise results below).

Our motivation comes partially from the notion of *distinct pair-sums* lattice polytopes [4, 7] (or dps polytopes, for short). A lattice polytope P is called *dps* if all the pairwise sums $a + b$, with $a, b \in P \cap \mathbb{Z}^d$, are distinct. Equivalently, if $P \cap \mathbb{Z}^d$ contains neither three collinear points nor the vertices of a non degenerate parallelogram ([4]). They are also the lattice polytopes of Minkowski length one, in the sense of [1].

An important fact is that dps d-polytopes have size at most 2^d ([4]). Hence, to classify all dps 3-polytopes, it would be enough to continue the work in this paper to a classification of sizes 7 and 8.

Our main result in this paper can be summarized as follows:

Theorem 2 *There are exactly 76 3-polytopes of size 6 and width > 1. 74 of them have width 2 and two have width 3. 44 and 1 of those, respectively, are dps.*

The volume vector (see definition in Section 2.1) and the width of each of these 76 polytopes is given in Tables 4 and 5. Different sections in the tables correspond to the presence or not of certain coplanarities (details in Section 2), as summarized in Table 1. Table 2 classifies the 76 polytopes according to the number of vertices and interior points. Following somehow established terminology we call a polytope *clean* if all its boundary lattice points are vertices, *hollow* if it has no interior lattice points, *canonical* if it has exactly one interior point and *terminal* if it is canonical and clean. In both tables, $x+1$ means that x polytopes have width two and one has width three.

Emptiness in the last line of the tables follows from the following result that Scarf attributes to Howe:

Theorem 3 ([8, Thm. 1.3]) *If all lattice points of a lattice 3-polytope are vertices then it has width 1.*

*This work is an extended abstract of [3], and is supported by grants MTM2011-22792 (both authors) and BES-2012-058920 (M. Blanco) of the Spanish Ministry of Science.

Department of Mathematics, Statistics and Computation, University of Cantabria, 39005 Santander, Spain

Description	# polys.	dps
$\exists$ 5 coplanar points	2	0
$\exists$ $(3,1)$ coplanarity, no 5 coplanar	$20+1$	13
$\exists$ $(2,2)$ coplanarity, none of above	4	0
$\exists$ $(2,1)$ coplanarity, none of above	17	0
No coplanarity, 1 interior point	20	20
No coplanarity, 2 interior points	$11+1$	$11+1$
No coplanarity, no interior points	0	0

Table 1: Number of lattice 3-polytopes of size 6 and width >1 according to coplanarities present in them.

#ver.	#int. pts.	Description	# polys.	dps
4	2	clean tetrahedra	$22+1$	$16+1$
	1	canonical tetrahedra	$10+1$	3
	0	hollow tetrahedra	2	0
5	1	terminal quad. pyramid	3	0
		terminal tri. bipyramid	35	24
	0	hollow quad. pyramid	1	0
		hollow tri. bipyramid	1	1
6	0	clean & hollow	0	0

Table 2: Number of lattice 3-polytopes of size 6 and width >1 according to the number of vertices and interior points.

Let us remark that clean tetrahedra and canonical 3-polytopes were previously classified:

- Kasprzyk has classified all canonical 3-polytopes ([6]). There are 674,688 of them, with their number of boundary lattice points going up to 38.
- Curcic classified clean tetrahedra with up to 35 interior points ([5]).

2 Preliminaries

2.1 Volume vectors and oriented matroids

Since $\mathbb{Z}$-equivalence preserves volume, the following *volume vector* is invariant under it:

Definition 1 *Let $A=\{p_1,p_2,\dots,p_n\}$, with $n\geq d+1$, be a set of lattice points in $\mathbb{Z}^d$. The* volume vector *of A is the vector $w=(w_{i_1\dots i_{d+1}})_{1\leq i_1<\dots<i_{d+1}\leq n}$ in $\mathbb{Z}^{\binom{n}{d+1}}$, where*

$$w_{i_1\dots i_{d+1}}:=\det\begin{pmatrix}1 & \dots & 1\\ p_{i_1} & \dots & p_{i_{d+1}}\end{pmatrix}$$

The definition implicitly assumes a specific ordering of the n points in A. For six points $\{p_1,\dots,p_6\}$ we always order the entries of the volume vector lexicographically as:

$$\begin{aligned} w \;=\; & (w_{1234},w_{1235},w_{1236},w_{1245},w_{1246},\\ & w_{1256},w_{1345},w_{1346},w_{1356},w_{1456},\\ & w_{2345},w_{2346},w_{2356},w_{2456},w_{3456})\end{aligned}$$

Theorem 4 ([2]) *Let A and B be d-dimensional subsets of $\mathbb{Z}^d$ with the same number n of points and suppose they have the same volume vector $(w_I)_{I\in\binom{[n]}{d+1}}$ with respect to a given ordering. Then:*

1. *There is a unique unimodular affine map $t:\mathbb{R}^d\to\mathbb{R}^d$ with $t(A)=B$ (respecting the order of points).*
2. *If $\gcd_{I\in\binom{[n]}{d+1}}(w_I)=1$, then t has integer coefficients, so it is a $\mathbb{Z}$-equivalence between A and B.*

That is: when its gcd equals 1, the volume vector is a *complete* invariant for $\mathbb{Z}$-equivalence.

(In particular, the volume vectors in Tables 4 and 5 are enough to recover representatives for each class, except in the five cases with gcd different from 1. Representatives of all classes can be found in [3]).

Observation 1 *The volume vector of $d+2$ points $\{p_1,\dots,p_{d+2}\}$ that affinely span $\mathbb{R}^d$ encodes the unique (modulo a scalar factor) affine dependence among them: let $I_k=\{1,\dots,d+2\}\setminus\{k\}$*

$$\sum_{k=1}^{d+2}(-1)^{k-1}\cdot w_{I_k}\cdot p_k=0,\qquad \sum_{k=1}^{d+2}(-1)^{k-1}\cdot w_{I_k}=0$$

Remember that the points with non-zero coefficient in this dependence form a circuit*, whose* signature *is the pair (i,j) if this dependence has i positive and j negative coefficients. We call* signature *of the $d+2$ points the signature of this circuit.*

In particular, the *oriented matroid* or *order type* of any point set A is encoded by the (signs in) the volume vector of A.

To classify lattice polytopes of size six we first classify the possible oriented matroids corresponding to an affine point configuration with 6 elements in dimension 3; that is, oriented matroids of *rank four* with *six elements* that are *acyclic* (i.e., without positive circuits), and that do not have any *parallel elements* (circuits of signature $(1,1)$). These are all representable, and there are 55 of them (see [3, Section 2.2]). A posteriori, it turns out that only 22 of them can be realized by the six lattice points in a lattice 3-polytope of size six.

2.2 Polytopes with 4 or 5 lattice points

We repeatedly use the classification of lattice 3-polytopes of size 4 and 5. Those of size four, or *empty tetrahedra*, were classified 50 years ago:

Theorem 5 (White [9]) *Every empty lattice tetrahedron is $\mathbb{Z}$-equivalent to the following $T(p,q)$, for some $q\in\mathbb{N}$ and $p\in\{0,q-1\}$ with $\gcd(p,q)=1$:*

$$T(p,q)=\operatorname{conv}\{(0,0,0),(1,0,0),(0,0,1),(p,q,1)\}$$

Moreover: (i) $q = \operatorname{vol} T(p,q)$ and (ii) $T(p,q)$ is equivalent to $T(p',q)$ if and only if $p' = \pm p^{\pm 1} \pmod q$.

Table 3 shows the full classification of 3-polytopes of size five, obtained in [2]. The polytopes are grouped according to the signature of their five lattice points. The five possible signatures are $(2,1)$, $(2,2)$, $(3,2)$, $(3,1)$ and $(4,1)$. We include the volume vector and width of each equivalence class. For convenience (see Observation 1), the volume vectors are written in the form $w = (w_{2345},\ -w_{1345},\ w_{1245},\ -w_{1235},\ w_{1234})$. Representatives of all classes can be found in [2].

Signature	Volume vector	Width
$(2,2)$	$(-1,1,1,-1,0)$	1
$(2,1)$	$(-2q,q,0,q,0)$, $0 \le p \le \frac{q}{2}$, $\gcd(p,q)=1$	1
$(3,2)$*	$(-a-b,a,b,1,-1)$, $0 < a \le b$, $\gcd(a,b)=1$	1
$(3,1)$*	$(-3,1,1,1,0)$	1
	$(-9,3,3,3,0)$	2
$(4,1)$*	$(-4,1,1,1,1)$	2
	$(-5,1,1,1,2)$	2
	$(-7,1,1,2,3)$	2
	$(-11,1,3,2,5)$	2
	$(-13,3,4,1,5)$	2
	$(-17,3,5,2,7)$	2
	$(-19,5,4,3,7)$	2
	$(-20,5,5,5,5)$	2

Table 3: Complete classification of lattice 3-polytopes of size 5. Those marked with an * are dps

3 Overview of the classification scheme

The proof of the classification involves a quite long case study. Let us here discuss the main ideas used in it. Let $A \subset \mathbb{Z}^3$ consist of six points, and assume that $\operatorname{conv}(A) \subset \mathbb{R}^3$ has size six and width greater than one. Then one of the following things occurs:

1. A contains 5 coplanar points. Then A consists of one of the six polygons of size five, plus an extra point at lattice distance at least two from it. It is not hard to show that only two of these six polygons allow this sixth point to be placed adding no additional lattice points in $\operatorname{conv}(A)$.

2. A contains a coplanarity of signature $(3,1)$ in a certain plane H (and no five coplanar points). This case is treated separately depending on whether the other two points of A lie in opposite or the same side of H. If they lie in opposite sides, then they are both at distance 1 or 3 of H, by the classification of size five. If they lie on the same side, then either they are both vertices of $\operatorname{conv}(A)$ (and lie at distance 1 or 3) or one is an interior point (at distance 1 or 3 from H) and the other is a vertex. In the cases where both are guaranteed to be at distance 1 or 3 we use what we call the *parallel-planes method* and when one is in the interior we use the $(4,1)$-*extension method* (see details below).

3. A contains a coplanarity of signature $(2,2)$ (and none of the above). This is treated in much the same way as the $(3,1)$ case, except things are now simpler because every time we said "distance 1 or 3" in the previous paragraph we can now say "distance 1".

4. All coplanarities in A come from collinearities of signature $(2,1)$. We first show that the $(2,1)$ collinearity must be unique (two of them would produce either width one or 5 coplanar points) and that removing from A one or the other extremal collinear points we get a configuration of signature $(4,1)$ and size 5. Once we have this we can use the $(4,1)$-*extension method*.

5. A is in general position (no coplanarities). A must have interior points, since otherwise it has width one by Theorem 3. There exist exactly two oriented matroids with this properties. It turns out that both oriented matroids have the following useful property: there are two vertices p_i and p_j of A such that both $A \setminus \{p_i\}$ and $A \setminus \{p_j\}$ have signature $(4,1)$. Configurations like these of size 6 can all be obtained by *gluing* two configurations of size 5 and signature $(4,1)$ along four points. There are only eight of these configurations that we need to consider (see Table 3) and (at most) $\left(\binom{8}{2}+8\right)\times 4^2 \times 4!$ possible ways to glue them, which we check one by one via computer routines written in MATLAB.

Let us explain the parallel planes and $(4,1)$-extension methods mentioned above:

- We use the *parallel-planes method* when we can guarantee that A is contained in three parallel planes H_1, H_2 and H_3 (not necessarily consecutive) and we know (or pose without loss of generality) the coordinates of all points but one. We look at what conditions must the coordinates of the sixth point satisfy for $\operatorname{conv}(A)$ not to have extra lattice points in H_2. This is a 2-dimensional problem that can be solved graphically. This gives us a finite (and small) list of possible positions for the unknown point, and it only remains to check that the size of $\operatorname{conv}(A)$ is indeed 6.

- We use the $(4,1)$-*extension method* when we know that there is a vertex p in A such that $A \setminus \{p\}$ has signature $(4,1)$ and, moreover, we know an expression of p as an affine combination of the other points. This happens when p is part of one of these relations: $3p_i = p_j + p_k + p_l$, $p_i + p_j = p_k + p_l$ or $2p_i = p_j + p_k$. We go through the $8 \times 4!$ possible ways to map $A \setminus \{p\}$ to one of the eight configurations of signature $(4,1)$ from Table 3, compute the corresponding p, and check whether the convex hull of the result has size six. This is done via some computer routines written in MATLAB.

We now list all the 3-polytopes of size six and width larger than one. They are contained in Tables 4 and 5, where we give, for each of them, the volume vector and the width. Configurations between horizontal lines correspond to the same oriented matroid. Dps ones are marked with an asterisk sign next to the width.

Volume vector	Width
Polytopes containing 5 coplanar points	
0 0 2 0 0 4 0 2 0 −4 0 4 −2 −8 −2	2
0 0 2 0 4 4 0 2 0 −4 0 0 −2 −4 −2	2
Polytopes containing a $(3,1)$ coplanarity (but no 5 coplanar points)	
0 1 −1 −1 1 −2 1 −1 0 2 3 −3 0 6 0	2
0 1 −1 −1 1 −1 1 −1 1 0 3 −3 0 3 −3	2
0 1 −1 −1 1 0 1 −1 0 0 3 −3 −2 2 −2	2
0 3 −3 −3 3 0 3 −3 0 0 9 −9 −6 6 −6	2
0 1 −1 −1 1 −4 1 −1 1 3 3 −3 3 9 0	2∗
0 1 −1 −1 1 −3 1 −1 0 3 3 −3 1 8 1	2∗
0 3 −3 −3 3 −9 3 −3 0 9 9 −9 3 24 3	2∗
0 1 −1 −1 1 −1 1 −1 0 1 3 −3 −1 4 −1	2
0 1 −1 −1 1 −5 1 −1 1 4 3 −3 4 11 1	2∗
0 1 −1 −1 1 −6 1 −1 1 5 3 −3 5 13 2	2∗
0 1 −3 −1 3 −8 1 −3 1 7 3 −9 5 19 2	2∗
0 1 −3 −1 3 −2 1 −3 1 1 3 −9 −1 7 −4	2∗
0 1 −1 −1 1 −2 1 −1 1 1 3 −3 1 5 −2	2∗
0 1 −1 −1 1 −3 1 −1 1 2 3 −3 2 7 −1	2∗
0 1 −3 −1 3 −5 1 −3 1 4 3 −9 2 13 −1	2∗
0 1 2 −1 −2 0 1 2 −1 1 3 6 0 0 3	2
0 1 2 −1 −2 0 1 2 0 0 3 6 1 −1 1	2
0 3 6 −3 −6 0 3 6 0 0 9 18 3 −3 3	**3**
0 1 5 −1 −5 1 1 5 −2 1 3 15 1 −4 7	2∗
0 1 7 −1 −7 1 1 7 −2 1 3 21 3 −6 9	2∗
0 1 3 −1 −3 −2 1 3 1 1 3 9 5 1 2	2∗
Polytopes containing a $(2,2)$ coplanarity (but none of the above)	
0 1 −1 1 −1 −4 −1 1 3 −1 −1 1 5 1 −2	2
0 1 −1 1 −1 −5 −1 1 4 −1 −1 1 7 2 −3	2
0 1 5 1 5 1 −1 −5 −2 −1 −1 −5 1 2 −3	2
0 1 7 1 7 2 −1 −7 −3 −1 −1 −7 1 3 −4	2
Polytopes containing a $(2,1)$ coplanarity (but none of the above)	
1 −1 −2 1 2 0 −1 −2 0 0 −4 −7 −1 1 −1	2
1 −2 −4 1 2 0 −1 −2 0 0 −5 −9 −2 1 −1	2
2 −1 −2 1 2 0 −1 −2 0 0 −5 −8 −1 1 −1	2
1 −3 −6 2 4 0 −1 −2 0 0 −7 −13 −3 2 −1	2
3 −2 −4 1 2 0 −1 −2 0 0 −7 −11 −2 1 −1	2
5 −3 −6 2 4 0 −1 −2 0 0 −11 −17 −3 2 −1	2
1 −1 1 1 −1 0 −1 1 0 0 −4 2 2 −2 2	2
1 −2 2 1 −1 0 −1 1 0 0 −5 3 4 −2 2	2
2 −1 1 1 −1 0 −1 1 0 0 −5 1 2 −2 2	2
1 −3 3 2 −2 0 −1 1 0 0 −7 5 6 −4 2	2
3 −2 2 1 −1 0 −1 1 0 0 −7 1 4 −2 2	2
5 −3 3 2 −2 0 −1 1 0 0 −11 1 6 −4 2	2
1 −1 −3 1 2 1 −1 −2 −1 0 −4 −8 −4 0 0	2
1 −1 −3 2 4 2 −1 −2 −1 0 −5 −10 −5 0 0	2
2 −1 −4 1 2 1 −1 −2 −1 0 −5 −10 −5 0 0	2
2 −1 −4 3 6 3 −1 −2 −1 0 −7 −14 −7 0 0	2
3 −1 −5 2 4 2 −1 −2 −1 0 −7 −14 −7 0 0	2

Table 4: Lattice 3-polytopes of size 6 and width > 1 (I)

Volume vector	Width
Polytopes with no coplanarities and 1 interior point	
1 −1 −1 1 3 −2 −1 −2 1 1 −4 −7 3 5 −1	2∗
1 −1 −3 1 5 −2 −1 −4 1 1 −4 −13 1 7 −3	2∗
1 −1 −1 1 2 −1 −2 −3 1 1 −5 −7 2 3 −1	2∗
1 −1 −2 1 3 −1 −2 −5 1 1 −5 −11 1 4 −3	2∗
1 −2 −5 1 4 −3 −1 −3 1 1 −5 −13 1 7 −2	2∗
2 −1 −1 1 5 −2 −1 −3 1 1 −5 −11 3 7 −2	2∗
2 −1 −3 1 7 −2 −1 −5 1 1 −5 −17 1 9 −4	2∗
1 −2 −3 1 2 −1 −3 −5 1 1 −7 −11 1 3 −2	2∗
2 −1 −1 1 3 −1 −3 −5 1 2 −7 −11 2 5 −1	2∗
2 −3 −7 1 5 −4 −1 −3 1 1 −7 −17 1 9 −2	2∗
3 −2 −1 1 5 −3 −1 −2 1 1 −7 −11 5 8 −1	2∗
3 −1 −1 1 4 −1 −2 −5 1 1 −7 −13 2 5 −3	2∗
3 −1 −2 1 5 −1 −2 −7 1 1 −7 −17 1 6 −5	2∗
3 −2 −5 1 7 −3 −1 −4 1 1 −7 −19 1 10 −3	2∗
5 −2 −1 1 3 −1 −3 −4 1 1 −11 −13 3 4 −1	2∗
5 −2 −3 1 4 −1 −3 −7 1 1 −11 −19 1 5 −4	2∗
5 −3 −5 1 5 −2 −2 −5 1 1 −11 −20 1 7 −3	2∗
3 −4 −5 1 2 −1 −5 −7 1 1 −13 −17 1 3 −2	2∗
4 −5 −7 1 3 −2 −3 −5 1 1 −13 −19 1 5 −2	2∗
5 −3 −4 1 3 −1 −4 −7 1 1 −13 −19 1 4 −3	2∗
Polytopes with no coplanarities and 2 interior points	
1 −1 5 1 1 −6 −1 −2 7 −1 −4 1 19 5 −9	2∗
1 −1 7 1 1 −8 −1 −2 9 −1 −4 3 25 7 −11	2∗
1 −2 5 1 1 −7 −1 −2 9 −1 −5 1 23 6 −11	2∗
1 −2 7 1 1 −9 −1 −2 11 −1 −5 3 29 8 −13	2∗
2 −1 7 1 1 −4 −1 −3 5 −1 −5 1 17 3 −8	2∗
2 −1 11 1 1 −6 −1 −3 7 −1 −5 5 25 5 −10	2∗
2 −3 7 1 1 −5 −1 −3 8 −1 −7 1 23 4 −11	2∗
2 −3 11 1 1 −7 −1 −3 10 −1 −7 5 31 6 −13	2∗
3 −1 7 2 1 −5 −1 −2 3 −1 −7 1 16 3 −5	2∗
3 −2 13 1 1 −5 −1 −4 7 −1 −7 5 27 4 −11	2∗
3 −5 11 2 1 −9 −1 −2 7 −1 −11 5 32 7 −9	2∗
5 −2 11 3 1 −7 −1 −2 3 −1 −11 3 23 4 −5	**3***

Table 5: Lattice 3-polytopes of size 6 and width > 1 (II)

References

[1] Olivia Beckwith, Matthew Grimm, Jenya Soprunova, Bradley Weaver. Minkowski length of 3D lattice polytopes. *Discrete Comput. Geom.* 48 (2012), 1137–1158.

[2] M. Blanco and F. Santos. Lattice 3-polytopes with five lattice points. Preprint, 19 pages, September 2014, `arXiv:1409.6701`.

[3] M. Blanco and F. Santos. Lattice 3-polytopes with six lattice points. Preprint, 31 pages, January 2015, `arXiv:1501.01055`.

[4] M. D. Choi, T. Y. Lam and B. Reznick. Lattice polytopes with distinct pair-sums. *Discrete Comput. Geom.* 27 (2002), 65–72.

[5] M. Curcic. Lattice polytopes with distinct pair-sums. Ph. D. Thesis, University of Illinois at Urbana-Champaign, 2012.

[6] A. Kasprzyk. Toric Fano 3-folds with terminal singularities. *Tohoku Math. J.* (2) 58 (2006), 101–121.

[7] B. Reznick. Problem 3 in "Let me tell you my favorite lattice-point problem" in *Integer Points in Polyhedra—Geometry, Number Theory, Representation Theory, Algebra, Optimization, Statistics*, M. Beck, C. Haase, B. Reznick, M. Vergne, V. Welker, and R. Yoshida (eds.). Contemp. Math. 452, Amer. Math. Soc. 2008.

[8] H. E. Scarf. Integral polyhedra in three space. *Math. Oper. Res.* 10:3 (1985), 403–438.

[9] G. K. White. Lattice tetrahedra. *Canadian J. Math.* 16 (1964), 389–396.

Automatic Proofs for Formulae Enumerating Proper Polycubes

Gill Barequet* Mira Shalah*

Abstract

In this paper we develop a general framework for computing formulae enumerating polycubes of size n which are proper in $n-k$ dimensions (i.e., spanning all $n-k$ dimensions), for a fixed value of k. (Such formulae are central in the literature of statistical physics in the study of percolation processes and collapse of branched polymers.) We re-affirm the already-proven formulae for $k \leq 3$, and prove rigorously, for the first time, that the number of polycubes of size n that are proper in $n-4$ dimensions is $2^{n-7}n^{n-9}(n-4)(8n^8-128n^7+828n^6-2930n^5+7404n^4-17523n^3+41527n^2-114302n+204960)/6$.

1 Introduction

A d-dimensional polycube of size n is a connected set of n cubes in d dimensions, where connectivity is through $(d-1)$-dimensional faces. Two *fixed* polycubes are considered the same if one can be obtained by a translation of the other. A polycube is said to be *proper* in d dimensions if the convex hull of the centers of its cubes is d-dimensional. Following Lunnon [7], we let $\mathrm{DX}(n,d)$ denote the number of fixed polycubes of size n that are proper in d dimensions. Similarly, we denote by $\mathrm{DT}(n,d)$ the number of tree polycubes of size n which are proper in d dimensions.

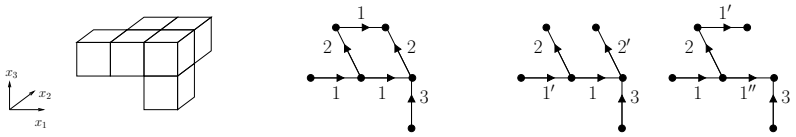
Enumeration of polycubes and computing their asymptotic growth rate are important problems in combinatorics and discrete geometry, originating in statistical physics [4]. While in the mathematical literature these objects are called polycubes (*polyominoes* in 2D), they are usually referred to as *lattice animals* in the literature of statistical physics, where they play a fundamental role in the analysis of percolation processes and collapse of branched polymers. To-date, no formula is known for $A_d(n)$, the number of fixed polycubes of size n in d dimensions, for any fixed value of d, let alone in the general case. The main interest in DX stems from the formula $A_d(n) = \sum_{i=0}^{d} \binom{d}{i} \mathrm{DX}(n,i)$, given originally by Lunnon [7]. In a matrix listing the values of DX, the top-right triangular half and the main diagonal contain only 0s. This gives rise to the question of whether a pattern can be found in the sequences $\mathrm{DX}(n,n-k)$, where $k < n$ is the ordinal number of the diagonal.

*Dept. of Computer Science, The Technion, Haifa, Israel. E-mail: {barequet,mshalah}@cs.technion.ac.il

Klarner [5] showed in a seminal work the existence of $\lambda_2 = \lim_{n\to\infty} \sqrt[n]{A_2(n)}$. Much later Madras [9] proved the convergence of the sequence $(A_2(n+1)/A_2(n))_{n=1}^{\infty}$ to λ_2 (and similarly in any fixed dimension d). Thus, λ_2 is the *growth rate* limit of polyominoes. Its exact value has remained elusive till these days. The currently best known lower and upper bounds on λ_2 are roughly 4.0025 [2] and 4.6496 [6], respectively. Significant progress in estimating λ_d has been obtained in statistical physics, although the computations usually relied on unproven assumptions and on formulae for $\mathrm{DX}(n,n-k)$ interpolated empirically from known values of $A_d(n)$. Peard and Gaunt [11] predicted that for $k > 1$, the diagonal formula $\mathrm{DX}(n,n-k)$ has the pattern $2^{n-2k+1}n^{n-2k-1}(n-k)h_k(n)$, where $h_k(n)$ is a polynomial in n, and conjectured explicit formulae for $h_k(n)$ for $k \leq 6$. Luther and Mertens [8] conjectured a formula for $k = 7$.

It is easy to show that $\mathrm{DX}(n,n-1) = 2^{n-1}n^{n-3}$ (sequence A127670 in OEIS [10]). Barequet et al. [3] proved rigorously, for the first time, that $\mathrm{DX}(n,n-2) = 2^{n-3}n^{n-5}(n-2)(2n^2-6n+9)$ (sequence A171860). The proof uses a case analysis of the possible structures of spanning trees of the polycubes, and the various ways in which cycles can be formed in their cell-adjacency graphs. Similarly, Asinowski et al. [1] proved that $\mathrm{DX}(n,n-3) = 2^{n-6}n^{n-7}(n-3)(12n^5-104n^4+360n^3-679n^2+1122n-1560)/3$, again, by counting spanning trees of polycubes, yet the reasoning and the calculations were significantly more involved. The inclusion-exclusion principle was applied in the proof in order to count correctly polycubes whose cell-adjacency graphs contained certain subgraphs, so-called "distinguished structures." In comparison with the case $k = 2$, the number of such structures is substantially higher, and the ways in which they can appear in spanning trees are much more varied. The latter proof provided a better understanding of the difficulties that one would face in applying this technique to higher values of k. The number of distinguished structures grows rapidly, and the inclusion relations between them are much more complicated. As anticipated [1], it is impractical to achieve a similar proof manually for $k > 3$.

In this paper we create a theoretical set-up for proving the formula for $\mathrm{DX}(n,n-k)$, for a fixed k. Using our implementation of this method, we find the explicit formula (which has never been proven before) for $\mathrm{DX}(n,n-4)$, stated in the following theorem.



(a) Polycube (b) Adjacency graph (c) Two spanning trees

Figure 1: A polycube P, the corresponding graph $\gamma(P)$, and spanning trees of $\gamma(P)$.

Theorem 1 $\mathrm{DX}(n, n-4) = 2^{n-7}n^{n-9}(n-4)(8n^8 - 128n^7+828n^6-2930n^5+7404n^4-17523n^3+41527n^2-114302n+204960)/6$.

2 Definitions and Notations

Integer Partition. A partition of a positive integer m is a way of writing m as the sum of one or more positive integers a_i, i.e., $m = a_1 + ... + a_h$. Two sums that differ only in the order of their summands are considered the same. Let $\Pi(m)$ denote the set of all partitions of m. We choose the canonic representation of a partition to be the list of its summands in nondecreasing order. For a partition p, we denote by $|p|$ the number of summands in p and by $p[i]$ the ith summand of p. Also, we let $\pi(p)$ denote the number of essentially-different permutations of the summands of p. When p_1 and p_2 are two partitions, we will say that p_1 contains p_2, denoting this relation by $p_2 \preceq p_1$, if there is a subpartition p_1^* of p_1 (an ordered subset of the elements of p_1), such that $|p_1^*| = |p_2|$ and $p_2[i] \leq p_1^*[i]$ for all $1 \leq i \leq |p_2|$.

Graph Isomorphism. Two directed edge-labeled graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, with respective edge labels W_G and W_H, are isomorphic if there is a bijection $f : V_G \leftrightarrow V_H$ s.t. (a) For $u, v \in V_G$ we have $(u, v) \in E_G$ if and only if $(f(u), f(v)) \in E_H$; and (b) For $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E_G$, the labels of e_1 and e_2 are equal if and only if the labels of $(f(u_1), f(v_1))$ and $(f(u_2), f(v_2))$ are equal.

3 Overview of the Method

Denote by $\mathcal{P}_n$ the set of proper polycubes of size n in $n-k$ dimensions. Let $P \in \mathcal{P}_n$, and let $\gamma(P)$ denote the directed edge-labeled graph that is constructed as follows: The vertices of $\gamma(P)$ correspond to the cells of P; two vertices of $\gamma(P)$ are connected by an edge if the corresponding cells of P are adjacent; an edge has label i ($1 \leq i \leq n-k$) if the corresponding cells have different i-coordinate. The direction of the edge is from the lower to the higher cell. See Figure 1 for an example. Since $P \mapsto \gamma(P)$ is an injection, it suffices to count the graphs obtained from the members of $\mathcal{P}_n$ in this way. We shall count these graphs by counting their spanning trees. A spanning tree of $\gamma(P)$ has $n-1$ edges labeled by numbers from the set $\{1, 2, ..., n-k\}$; all these labels are present, otherwise, the polycube is not proper in $n-k$ dimensions. Hence, $n-k$ edges of the spanning are labeled with the labels $1, 2, ..., n-k$, and the remaining $k-1$ edges are labeled with repeated labels from the same set. There is a bijection between the possibilities of repeated edge-labels and the partitions of the integer $k-1$. Specifically, each partition $p = \sum_{i=1}^{h} a_i \in \Pi(k-1)$ corresponds to h repeated labels in the spanning tree, such that the ith repeated label appears a_i+1 times. In such case, we will say that the tree is *labeled according to* p. When we consider a spanning tree of $\gamma(P)$, we distinguish a repeated label i that appears r times by $i, i', ..., i'^{(r-1)}$. However, when considering $\gamma(P)$, repeated labels are assumed not to be distinguished. Every repeated label must occur an even number of times in any cycle of $\gamma(P)$. In addition, the number of cycles in $\gamma(P)$ and the length of each such cycle are bounded from above due to the limited multiplicity of labels.

In order to compute $|\mathcal{P}_n|$, we go over all possible directed edge-labeled trees of size n and count only those that represent valid polycubes. By characterizing and analyzing all substructures that are present in these trees, we compute how many of them actually represent polycubes. Finally, we develop formulae for the numbers of possible spanning trees of polycubes, and then derive the actual number of polycubes.

4 Distinguished Structures

For each labeled tree, we attempt to build the corresponding polycube. In this process two things may happen:

(a) Cells may coincide (Figures 2(a,d)). A tree with overlapping cells is invalid and does not correspond to a valid polycube; and

(b) Two cells which are not connected by a tree edge may be adjacent (Figures 2(b,e)). Such a tree corresponds to a polycube P with cycles in $\gamma(P)$, and therefore, its spanning tree is not unique.

In order to count correctly, we will consider small structures, contained in these trees, which cause the problems above. A *distinguished structure* is defined as the union of all paths (edges and incident vertices) that run between two coinciding or adjacent cells. We

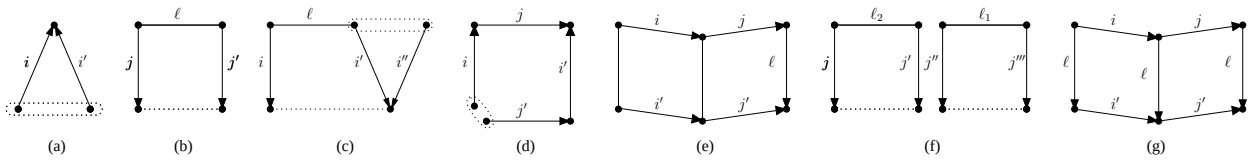


Figure 2: (a–f) A few distinguished structures for $k = 4$ (note that (f) is disconnected); (g) A cycle structure. A dotted line is drawn between every pair of neighboring cells and around every pair of coinciding cells.

denote by $\mathcal{DS}_k$ the set of distinguished structures in $n-k$ dimensions. With this characterization of distinguished structures, it is easy to design an algorithm for producing $\mathcal{DS}_k$. We begin by generating all "free trees" (non-isomorphic trees) of size (number of vertices) $t \leq m(k)$, where $m(k)$ is a function of k specified below. Then, we process each free tree T by labeling it according to every partition $p \in \cup_{i=1}^{k-1}\Pi(i)$, directing its edges for obtaining a directed edge-labeled tree T', and finally checking whether T' contains coinciding or neighboring cells by a DFS traversal. If such cells are detected, T' is added to $\mathcal{DS}_k$ if it is *not* isomorphic to any structure $\sigma \in \mathcal{DS}_k$ of size t, and at least one of the following conditions holds:

1. T' contains two coinciding or neighboring cells which are connected by a path with $t-1$ edges (Figures 2(a,b,d,e));
2. T' is isomorphic to the union of $d_1, ..., d_m \in \mathcal{DS}_k$, such that the isomorphic copies of $d_1, ..., d_m$ in T' cover all its edges (Figure 2(c)).

Disconnected distinguished structures (see Fig. 2(f)) are generated by checking if every collection of edge-connected structures in $\mathcal{DS}_k$ yields a single disconnected structure labeled according to $p \in \cup_{i=1}^{k-1}\Pi(i)$.

Lemma 2 *A connected (resp., disconnected) distinguished structure can have at most $3k-2$ (resp., $4k$) vertices. These sizes are attainable in the worst case.*

Lemma 3 *[1, Lemma 7] [3, Lemma 2] The number of directed trees with n vertices and $n-1$ distinct edge labels $1, ..., n-1$ is $2^{n-1}n^{n-3}$, for $n \geq 2$.*

Let T_p denote the number of directed trees with n vertices labeled according to $p \in \Pi(k-1)$. Then, $T_p = \pi(p)\binom{n-k}{|p|}2^{n-1}n^{n-3}$.

Lemma 4 *Let σ be a distinguished structure composed of $k^* \geq 1$ trees $s_1, ..., s_{k^*}$ with a total of n^* vertices and distinct edge labels $1, ..., n^* - k^*$. The number of occurrences of σ in trees of size n with distinct edge labels $1, ..., n-1$ is*

$$F_n(\sigma) = (\textstyle\prod_{i=1}^{k^*} |s_i|)\frac{(n-n^*+k^*-1)!}{(n-n^*)!}n^{n-n^*+k^*-2}.$$

Proof. We proceed by double counting, enumerating the sequences of directed edges that can be added to a graph with $n-n^*$ vertices and the distinguished structure σ, so as to form a rooted tree with n vertices. One way is to add the edges one by one and count the number of options available at each step. There are $\mathcal{N} = \prod_{i=1}^{k^*}|s_i|$ options to choose a root for each component s_i of σ. At the beginning, we have a forest with $n-n^*+k^*$ rooted trees. After adding a collection of edges forming a rooted forest with i trees, there are $n(i-1)$ choices for the next edge: Its starting vertex can be any of the n vertices of the graph, and its ending vertex can be any of the $i-1$ roots other than the root of the tree containing the starting vertex. Therefore, the total number of choices is

$$\mathcal{N}\prod_{i=2}^{n-n^*+k^*} n(i-1) = \mathcal{N}n^{n-n^*+k^*-1}(n-n^*+k^*-1)!. \quad (1)$$

Alternatively, start with an unrooted edge-labeled tree which contains σ, choose one of its n vertices as a root, and choose one of the $(n-n^*)!$ possible sequences, say, η, then label the $n-n^*$ vertices of the tree according to η (the vertices not belonging to σ), and "shift" each vertex label to the incident edge towards the root, producing an edge-labeled tree. The total number of sequences that are formed this way is

$$nF_n(\sigma)(n-n^*)!. \quad (2)$$

It follows from Eqs. (1) and (2 that the number of occurrences of σ in unrooted trees with edge labels $1, ..., n-1$ is $F_n(\sigma) = \mathcal{N}\frac{(n-n^*+k^*-1)!}{(n-n^*)!}n^{n-n^*+k^*-2}$. □

Consider again Fig. 2. Let $\mathcal{F}_n(\sigma)$ denote the number of occurrences of σ in *directed* edge-labeled trees of size n. Obviously, $\mathcal{F}_n(\sigma) = 2^{n-n^*+k^*-1}F_n(\sigma)$. Let $\sigma^* \in \mathcal{DS}_k$ be a distinguished structure labeled according to $p^* \in \cup_{i=1}^{k-1}\Pi(i)$. Denote by $\mathcal{O}_p(\sigma^*)$ the number of occurrences of σ^* in directed trees of size n that are labeled according to $p \in \Pi(k-1)$. Clearly, if $p^* \not\trianglelefteq p$, then $O_p(\sigma^*) = 0$. To compute $\mathcal{O}_p(\sigma^*)$, one has to choose $|p|$ repeated labels out of the possible $n-k$ labels, then choose the repeated labels of σ^* out of the $|p|$ repeated labels. One also has to choose the unique labels (if any) in σ^*. Moreover, one has to calculate how many essentially-different structures there are out of all $\prod_{j=1}^{|p^*|}(p^*[j]!)$ possible configurations of the repeated labels of σ^*. For structure (a),

all configurations yield the same structure, whereas for structure (b), there are two essentially-different structures: in the first, the label i is attached to the head of the edge labeled ℓ; and in the second, i' is attached to its head. For structure (c) there are six different structures, a number obtained by computing the number of symmetries of σ^*. Finally, multiplying by $\mathcal{F}_n(\sigma^*)$ completes the calculation of $\mathcal{O}_p(\sigma^*)$.

When counting the occurrences of $\sigma \in \mathcal{DS}_k$, other distinguished structures which contain multiple occurrences of σ are counted multiple times. In order to obtain the number of *trees* that contain σ, using the quantity $\mathcal{O}_p(\sigma)$, we build an inclusion-exclusion graph IE=($\mathcal{V}$,$\mathcal{E}$). This graph contains a vertex corresponding to each structure $\sigma \in \mathcal{DS}_k$. There is an edge $e = \sigma_1 \to \sigma_2$ labeled with c if σ_1 contains c occurrences of σ_2. Let $\ell(e)$ denote the label of the edge e. Let $I(\sigma_2) = \{\sigma_1 \in \mathcal{V} : (\sigma_1, \sigma_2) \in \mathcal{E}\}$. Let us denote by $T_p(\sigma)$ the number of trees of size n labeled according to $p \in \Pi(k-1)$ that contain σ as a subtree, then, $T_p(\sigma_2) = \mathcal{O}_p(\sigma_2) - \sum_{\sigma_1 \in I(\sigma_2)} \ell((\sigma_1, \sigma_2)) T_p(\sigma_1)$.

5 Counting Polycubes

Every tree polycube gives rise to a unique spanning tree. For every possibility of repeated labels $p \in \Pi(k-1)$, let us denote by $\mathrm{DT}_p(n)$ the number of spanning trees of all tree polycubes that are labeled according to p. The total number of directed trees with n vertices labeled according to p is T_p. Every such tree corresponds to a tree polycube in $\mathcal{P}_n$ unless it contains a distinguished structure as a subtree. Thus, we need to exclude all the trees that contain a distinguished structure as a subtree. Hence,

$$\mathrm{DT}(n, n-k) = \sum_{p \in \Pi(k-1)} \mathrm{DT}_p(n)$$
$$= \sum_{p \in \Pi(k-1)} \frac{T_p - \sum_{\sigma \in \mathcal{DS}_k} T_p(\sigma)}{\prod_{j=1}^{|p|} p[j]!}.$$

Let us denote by $\mathcal{C} = \mathcal{C}(k)$ the set of all cycle structures of polycubes proper in $n-k$ dimensions. The set $\mathcal{C}$ can be found using $\mathcal{DS}_k$: A distinguished structure is a spanning tree of a cycle if it contains only neighboring cells and no coinciding cells. For example, in Figure 2, structure (e) is a spanning tree of the cycle (g). For every cycle structure $\mathcal{C}_i \in \mathcal{C}$, let us denote by $P_{\mathcal{C}_i}$ the number of polycubes $P \in \mathcal{P}_n$ that contain $\mathcal{C}_i$ in $\gamma(P)$. Let $\sigma \in \mathcal{DS}_k$ have c occurrences in $\mathcal{C}_i$. Then,

$$P_{\mathcal{C}_i} = \sum_{p \in \Pi(k-1)} \frac{T_p(\sigma)}{c}.$$

Finally, we obtain

$$\mathrm{DX}(n, n-k) = \mathrm{DT}(n, n-k) + \sum_{i=0}^{|\mathcal{C}(k)|-1} P_{\mathcal{C}_i}.$$

6 Results

The entire method was automated in a C++ program, using *Mathematica* to simplify the final formula. Our results agree with the formulae conjectured in the literature of statistical physics. For k=3, 147 distinguished structures (137 connected) and 12 types of cycles were found. For k=4, 8397 distinguished structures (7814 connected), and 179 cycles were found. Here are the main steps of the computation.

$\mathrm{DT}_{(2,2,2)}(n) = 2^{n-7}(n-4)(n-5)(n-6)n^{n-9}(8n^6 - 84n^5 + 438n^4 - 1543n^3 + 4236n^2 - 9020n + 19040)/6$

$\mathrm{DT}_{(2,3)}(n) = 2^{n-4}(n-4)(n-5)n^{n-9}(4n^6 - 56n^5 + 383n^4 - 1654n^3 + 5106n^2 - 10920n + 14112)/6$

$\mathrm{DT}_{(4)}(n) = 2^{n-5}(n-4)n^{n-9}(4n^6 - 84n^5 + 851n^4 - 5191n^3 + 20190n^2 - 47552n + 53760)/6$

$\mathrm{DT}(n, n-4) = \mathrm{DT}_{(2,2,2)}(n) + \mathrm{DT}_{(2,3)}(n) + \mathrm{DT}_{(4)}(n) = 2^{n-7}(n-4)n^{n-9}(8n^8 - 140n^7 + 1010n^6 - 3913n^5 + 9201n^4 - 15662n^3 + 34500n^2 - 120552n + 221760)/6$

$\sum_{i=0}^{178} P_{\mathcal{C}_i} = 2^{n-7}(n-4)(n-5)n^{n-9}(12n^6 - 122n^5 + 373n^4 + 68n^3 - 1521n^2 - 578n + 3360)/6$

$\mathrm{DX}(n, n-4) = \mathrm{DT}(n, n-4) + \sum_{i=0}^{178} P_{\mathcal{C}_i} = 2^{n-7}n^{n-9}(n-4)(8n^8 - 128n^7 + 828n^6 - 2930n^5 + 7404n^4 - 17523n^3 + 41527n^2 - 114302n + 204960)/6$

The parallel computation took about 15 minutes on a computer with 12 processors and 65 GB of RAM. This completes the proof of Theorem 1.

References

[1] A. Asinowski, G. Barequet, R. Barequet, and G. Rote, Proper n-cell polycubes in $n-3$ dimensions, *J. of Integer Sequences*, 15 (2012), #12.8.4.

[2] G. Barequet, G. Rote, and M. Shalah, $\lambda > 4$, 30^{th} European Workshop on Computational Geometry, Ein-Gedi, Israel, March 2014.

[3] R. Barequet, G. Barequet, and G. Rote, Formulae and growth rates of high-dimensional polycubes, *Combinatorica*, 30 (2010), 257–275.

[4] S.R. Broadbent and J.M. Hammersley, Percolation processes: I. Crystals and mazes, *Proc. Cambridge Philosophical Society*, 53 (1957), 629–641.

[5] D.A. Klarner, Cell growth problems, *Canadian J. of Mathematics*, 19 (1967), 851–863.

[6] 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 (1973), 585–602.

[7] W. F. Lunnon, Counting multidimensional polyominoes, *The Computer Journal*, 18 (1975), 366–367.

[8] S. Luther and S. Mertens, Counting lattice animals in high dimensions, *J. of Statistical Mechanics: Theory and Experiment*, 9 (2011), 546–565.

[9] N. Madras, A pattern theorem for lattice clusters, *Annals of Combinatorics*, 3 (1999), 357–384.

[10] *OEIS*, available at `http://oeis.org`

[11] P.J. Peard and D.S. Gaunt, $1/d$-expansions for the free energy of lattice animal models of a self-interacting branched polymer, *J. Physics, A: Mathematical and General*, 28 (1995), 6109–6124.

Compact families of Jordan curves and convex hulls in three dimensions (extended abstract)

Colm Ó Dúnlaing*

Abstract

We prove that for certain families of semi-algebraic convex bodies in $\mathbb{R}^3$, the convex hull of n disjoint bodies has $O(n\lambda_s(tn))$ features, where s and t are constants depending on the family: $\lambda_s(n)$ is the maximum length of order-s Davenport-Schinzel sequences with n letters. The argument is based on an apparently new idea of 'compact family' of convex bodies or discs, and of 'crossing content' among disc intersections.

1 Introduction

Let S be a set of n disjoint closed bounded convex bodies in $\mathbb{R}^3$: $H(S)$ is their convex hull, the smallest closed convex set containing their union $\bigcup S$.

$H(S)$ can be decomposed into features, consisting of facets (exposed facets, tunnel facets, and planar facets [2]), edges, and vertices; facets meet along edges: edges meet at vertices. $H(S)$ has $O(e)$ features (feature complexity), where e is the number of edges.

This abstract is concerned with the feature complexity of $H(S)$ assuming that the bodies in S are drawn from a 'compact family' of convex bodies, a term defined later. The feature complexity of convex hulls and Voronoi diagrams of point sites in $\mathbb{R}^3$ is well known, $O(n)$ and $O(n^2)$ respectively; and also for spherical bodies ($O(n^2)$ for both). Hung and Ierardi [3] studied the convex hull $H(S)$ where S is a set of bodies with few restrictions. They reported upper bounds on the feature complexity of convex hulls, but their approach is indirect and hard to follow. Our approach stresses a more careful definition of the problem (in terms of compact families): the feature complexity of $H(S)$ is reduced to the feature complexity (counting edges and vertices) of unions of closed discs on the boundaries ∂B of the bodies B in S.

After highlighting the main points in the analysis, the abstract gives some time to bounding feature complexity given a property called *positive crossing content,* defined below. An indication of why this property holds for compact families of discs is given (Section 4). The continuum mathematics connecting compact families of convex bodies (Section 5) to compact families of discs has been omitted for reasons of space.

*Mathematics, Trinity College, Dublin 2, Ireland. E-mail odunlain@maths.tcd.ie

2 Outline of the analysis

Definition 1 *If X is a set in a topological space then $\overline{X}$ is the closure of a set X, X° its interior, and ∂X its boundary, i.e., $\overline{X}\backslash X^\circ$.*

S^2 is the unit sphere $\{x \in \mathbb{R}^3 : \|x\| = 1\}$.

A compact convex body B is rounded *if (i) for every point x on ∂B, there is a* unique *tangent plane to B at x, and hence a* unique *outward unit normal $n_B(x)$ to B at x, and (ii) $x \neq y \Rightarrow n_B(x) \neq n_B(y)$. Equivalently, (ii) every tangent plane meets B at exactly one point.*

We call the map $n_B : \partial B \to S^2$ the outward normal map.

Proposition 1 [2]. *If B is a rounded compact convex body then n_B is a homeomorphism.*

The chief points in the analysis are as follows.

- $S = \{B_1, \ldots, B_n\}$ are disjoint rounded compact convex bodies.
- Given any body B_i, the *hidden* and *exposed regions* of B_i (relative to $H(S)$) are, respectively, the closed subsets

$$\overline{(\partial B_i) \cap H(S)^\circ} \quad \text{and} \quad \partial B_i \backslash H(S)^\circ.$$

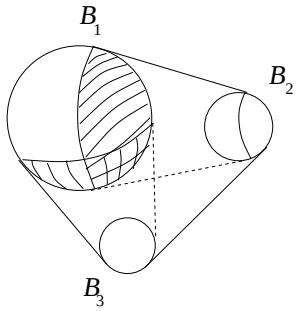


Figure 1: *The hidden part of ∂B_1 is hatched. It is the union of two topological discs.*

- Given two (disjoint) bodies B_i, B_j, B_j induces a hidden region on ∂B_i. This region, and the exposed region, are topological discs, separated by

a Jordan curve on ∂B_i, which we call the B_i, B_j-*seam*.

- The features of $H(S)$ incident to B_i correspond to the edges and vertices in the exposed (and hidden) regions on ∂B_i. Thus the feature complexity of $H(S)$ reduces to the complexity of unions of discs on ∂B_i.

- Given a seam on B_i, its image under n_{B_i} is a Jordan curve on S^2, a 'pre-seam' (Proposition 1), allowing us to study unions of discs on a fixed space S^2.

- Certain assumptions on the bodies B_i will imply that the pre-seams are continuously differentiable and the family of pre-seams is compact with respect to the C^1-norm (Definition 5).

  We then say that the pre-seams form a *compact family* of Jordan curves on S^2.

- The definition of compact family introduces, implicitly, a novel idea of 'fatness' on the discs which the family defines: it might be called 'stiffness.' This leads to the all-important property of *positive crossing content* (Lemma 2) from which the complexity bounds on $H(S)$ follow.

- Two topological closed discs D_i and D_j on S^2 (with C^1 boundaries) are *in general position* if there are finitely many points in $\partial D_i \cap \partial D_j$ and at any such point the tangents to D_i and D_j meet transversely.

  A list $D_1, \ldots, D_n$ of discs is in general position if every two discs are in general position and for any three different discs, $\partial D_i \cap \partial D_j \cap \partial D_k = \emptyset$.

- An *intersection component* is a connected component of $D_i \cap D_j$, the intersection of two discs. When in general position, every component is bounded by sides alternately from ∂D_i and ∂D_j. When there are two sides, we call the component an *overlap*, and otherwise a *crossway*.

- A family of discs in S^2 has *bounded intersection* if there is a uniform upper bound on $|\partial D_i \cap \partial D_j|$ for any two discs in general position.

- Overlaps can be shown to contribute $O(n)$ vertices to the (external) boundary of a union of n discs, given bounded intersection and positive crossing content (Corollary 5). Without positive crossing content there can be $\Omega(n^2)$ crossways (Figure 3).

Lemma 2 (positive crossing content). *Given a compact family of Jordan curves on S^2, there exists a positive lower bound, whose supremum we call the* crossing content *of the family, on*

$$\mu(K)$$

the metric measure (in S^2) of any crossway K between discs in general position.

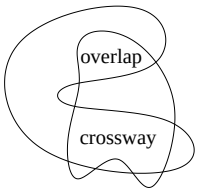


Figure 2: *overlap and crossway.*

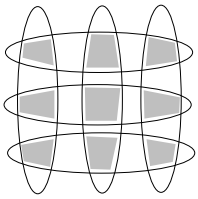


Figure 3: *a grid of n discs with $\Omega(n^2)$ crossways.*

- There is a combinatorial argument showing the following. Let $D_1, \ldots, D_n$ be discs in general position in S^2, with bounded intersection and positive crossing content, and connected union, so the complement of the union has simply-connected components H_i.

  Then there are $O(n)$ pairs (H_i, D_j) where D_j has an edge in common with H_i (Lemma 7).

Our estimate of the feature complexity of $\bigcup D_j$ and of $H(S)$ follows immediately (Corollaries 8 and 10).

3 Complexity of unions of discs with positive crossing content

Let $S = \{D_1, \ldots, D_n\}$ be a set of discs, with C^1 boundaries, in general position in S^2. We consider the feature complexity of their union, $\bigcup S$. We assume that the discs D_i come from a family with bounded intersection number and positive crossing content ($\geq \epsilon$, say).

Definition 2 *Let U be the union of all crossways between pairs of discs D_i, D_j, $1 \leq i < j \leq n$. A* hub *is a (path-) connected component of U.*

Lemma 3 *There are $O(1)$ hubs.*

Proof. Every hub contains at least one crossway, and therefore has measure $\geq \epsilon$. Hubs are disjoint, and $\mu(S^2) = 4\pi$, so there are at most $4\pi/\epsilon$ hubs. □

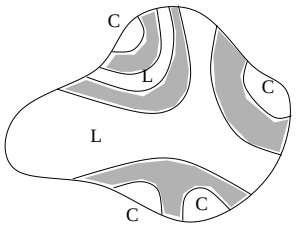


Figure 4: *connected unions of crossways in D_i, links, and coves. Overlaps are ignored.*

Definition 3 *Suppose that D_i is a disc and U is the union of crossways: by the arguments in Lemma 3, $D_i \cap U$ has $O(1)$ components. Different components may be part of the same hub. Let $K_1, \ldots, K_k$ be these components. In Figure 4 they are shaded. (Only crossways are considered here; possible overlaps are omitted from the figure.)*

$$X = D_i \backslash K_1 \ldots \backslash K_k$$

has a potentially unbounded number of components, but if we distinguish links *from* coves *there is a bounded number of links.*

A link *in D_i is either D_i itself, if $k = 0$ (D_i has no crossways), or it is (the closure of) a component of the above subset X of D_i whose intersection with ∂D_i is nonempty and disconnected.*

A cove *is a component whose intersection with ∂D_i is nonempty and connected.*

An external link segment *is a connected component of $L \cap \partial D_i$, where L is a link in D_i.*

Lemma 4 *In each disc D_i there are $O(1)$ links and external link segments.*[1]

Proof. From the links one can derive a recursive partition of the set $K_1, \ldots, K_k$, a tree structure in which there are $\leq 2k$ internal nodes, matching the external link segments: but k is bounded. □

Corollary 5 *There are $O(n)$ overlaps.*

Proof. There are $O(n)$ links and hubs; retract the overlaps to become very thin; choose centres in all links and hubs; one can thread paths joining centres through the overlaps; because of bounded intersection and planarity considerations, there are $O(n)$ overlaps. □

The overlaps can be further retracted until they become empty, making all overlaps disappear. This reduces the number of features by $O(n)$, and from now on we **assume there are no overlaps, only crossways.**

[1] There can be arbitrarily many coves.

Definition 4 *We may assume $\bigcup D_i$ is connected, since its feature complexity is the sum of those of its connected components.*

A hole *is the closure of a connected component of $S^2 \backslash \bigcup D_i$.*

Since the union is connected, every hole is simply connected.

Lemma 6 *Combinatorial lemma: if $\bigcup_1^n D_i$ is connected, then by re-ordering the list $D_1, \ldots, D_n$ if necessary, it can be arranged that every partial union $\bigcup_1^k D_i$, $1 \leq k \leq n$, is connected.*

Lemma 7 *There are $O(n)$ pairs D_i, H_j where H_j is a hole incident to D_i.*

Proof. We can assume that $\bigcup_1^k D_i$ is connected for $1 \leq k \leq n$. We apply (forward) induction on k. Suppose the disc D_k is added to an existing union $\bigcup_1^{k-1} D_i$ ($k \geq 2$). It is enough to show that $O(1)$ new holes are created.

The number of holes is increased by virtue of an existing hole, or holes, H, being split into several, $H_1, \ldots, H_\ell$, by D_k. The holes are always simply connected.

Let H_r and H_s be holes, part of the same hole H split by D_k. H is (simply) connected. Consider any path in H joining points y_r and y_s interior to H_r and H_s. The path crosses ∂D_k at least twice. If the path wanders into a cove from H_r, it must wander out again without leaving H_r. So the path must cross some external link segment incident to H_r. Thus all the holes H_r are incident to external link segments in D_k: there are $O(1)$ external link segments, so adding the disc D_k creates $O(1)$ new holes. □

Corollary 8 *There exist constants s and t such that $\bigcup D_i$ has feature complexity $O(\lambda_s(tn))$.*

Proof. For any H_i, suppose there are t_i discs D_j sharing an edge with H_i; $\sum_i t_i \leq tn$ for some constant t.

Let $e_1, \ldots, e_k$ be the edges incident to H_i, in anticlockwise order; each edge is on one of the discs D_j, and if we take the corresponding list of discs D_j, with repetitions, since the discs have bounded intersection number, we have a Davenport-Schinzel sequence. Therefore for some constant s, H_i has $\leq \lambda_s(t_i)$ edges. Adding, $\bigcup D_j$ has $O(\lambda_s(tn))$ features. □

4 Compact families of discs

The obvious metric connecting compact sets is the *Hausdorff distance* whose definition we omit to save space. We refer to [1] for important facts about the Hausdorff metric.

We need a finer metric on the space of closed discs in S^2.

The definition of 'compact family of discs' in S^2 is chosen to ensure that every such family has positive crossing content. It is reasonable to parametrise the discs D by a mapping $f : [0, 2\pi] \to S^2$ so that ∂D is the image of f, a Jordan curve, with D to its left.

As noted before, the maps f should be semialgebraic of bounded degree, to ensure bounded intersection.

Clearly we can take two triangles in the plane and move them around so that they have a crossway of arbitrarily small area. This is because the triangle boundaries are not differentiable. So the maps f should be continuously differentiable.

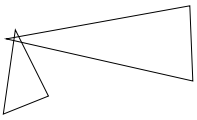


Figure 5: *positive crossing content needs differentiable boundaries.*

Definition 5 *The C^1 metric on C^1 maps: $[0, 2\pi] \to S^2$ is*

$$\max\left(\sup_{0\le t\le 2\pi} \|f(t) - g(t)\|, \sup_{0\le t\le 2\pi} \|f'(t) - g'(t)\|\right).$$

A family $\mathcal{F}$ of discs, parametrised by continuously differentiable maps f, is compact *if it is a compact space under the C^1 metric.*

The crux of the proof that compact families of discs have positive crossing content (Lemma 2) is: if K is a connected component of $D \cap E$ and $\mu(K) = 0$, then K must be a subinterval of $\partial D \cap \partial E$, and D and E must be locally on opposite sides of K. If we perturb D and E slightly, we can only introduce overlaps near K, not crossways.

5 Compact families of convex bodies

For reasons of space, the remainder is very sketchy. We consider a family $\mathcal{G}$ of convex bodies in $\mathbb{R}^3$. Such bodies are parametrised by maps f, but implicitly rather than explicitly: that is, each body should be described as

$$B_f = \{x : \ f(x) \le 1\}$$

where f is a C^2 function with nonzero first derivative and positive definite second derivative and bounded algebraic complexity. There is a C^2 metric on such functions f. The family $\mathcal{G}$ is *compact* if it is compact under the C^2 metric.

Proposition 9 *Let Π be the family of pre-seams from disjoint or tangential pairs B_1, B_2 drawn from $\mathcal{G}$. The pre-seam map is continuous on these pairs and the discs the pre-seams bound is a compact family.*

Proof. Laborious: see [5]. □

There is an immediate corollary.

Corollary 10 *Let S be a set of n disjoint convex bodies in $\mathbb{R}^3$ drawn from a compact family $\mathcal{G}$. Then there exist constants s and t so that the convex hull $H(S)$ has $O(n\lambda_s(tn))$ features.*

6 The Davenport-Schinzel effect

While it is doubtful that Corollary 10 is sharp, nonlinear effects are unavoidable when positive crossing content is invoked. Superlinear complexity of the lower envelopes of line-segments (see, for instance, [4]) carries over to polygonal discs with positive crossing content, as Figure 6 suggests.

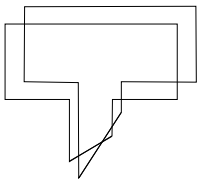


Figure 6: *nonlinear effect with arbitrarily large crossing content.*

Acknowledgments

The author thanks the anonymous referees for their comments and has tried to follow them in revising the draft.

References

[1] Jeff Henrikson (1999). Completeness and total boundedness of the Hausdorff metric. *MIT Undergraduate Journal of Mathematics.*

[2] Paul Harrington, Colm Ó Dúnlaing, and Chee-Keng Yap (2007). Optimal Voronoi diagram construction with n convex sites in three dimensions. *International Journal of Computational Geometry and Applications,* **17:6**, 555–593.

[3] C.-K. Hung and D. Ierardi (1995). Constructing convex hulls of quadratic surface patches. *Proc 7th Canadian Conf. on Computational Geometry*, 255–260.

[4] Jiří Matoušek (2002). *Lectures on discrete geometry.* Springer GTM 212.

[5] Colm Ó Dúnlaing (2013). Compact families of Jordan curves and convex hulls in three dimensions. TCD-MATH 13–15, arXiv 1311.6331. Submitted for publication.

New Geometric Algorithms for Staged Self-Assembly

Erik D. Demaine* Sándor P. Fekete† Arne Schmidt†

Abstract

We consider *staged self-assembly*, in which square-shaped Wang tiles can be added to bins in several stages. Within these bins the tiles may connect to each other, depending on the *glue types* of their edges. In general, self-assembly constructs complex (polyomino-shaped) structures from a limited set of square tiles. Previous work by Demaine et al. considered a setting in which assembly proceeds in stages. It was shown that a relatively small number of tile types suffices to produce arbitrary shapes; however, these constructions were only based on a spanning tree of the geometric shape, so they did not produce full connectivity of the underlying grid graph. We present new systems for stages assembly to assemble a fully connected polyomino in $O(\log^2 n)$ stages. Our construction works even for shapes with holes and uses only a constant number of glues and tiles.

1 Introduction

In *self-assembly*, a set of simple *tiles* form complex structures without any active or deliberate handling of individual components. Instead, the overall construction is governed by a simple set of rules, which describe how mixing the tiles leads to bonding between them and eventually a geometric shape.

The leading theoretical model for self-assembly is the *abstract tile-assembly model* (aTAM). It was first introduced by Winfree [12, 9]. The *tiles* used in this model are building blocks called *Wang tiles* [11], which are unrotatable squares with a specific glue on each side. Equal glues have a connection strength and may stick together. If two partial assemblies ("supertiles") want to assemble, then the sum of the glue strength along the linkage needs to be at least some minimum value τ, which is called the *temperature*. While assembling some shapes we consider the minimum number of distinct tiles to uniquely assemble this shape; this is called the *tile complexitiy* t. Apart from that we also consider a minimum number of glues, which is the *glue complexitiy* g. Clearly, $g \leq t \leq g^4$.

In this paper we consider the *staged tile assembly model* introduced by Demaine et al. [3]. In this model the assembly process is split into sequential stages that are kept in separate bins, with supertiles from earlier stages mixed together consecutively to gain some new supertiles. We can either add a new tile to an existing bin, or we pour one bin into another bin, such that the content of both get mixed. Hence, there are bins at each stage. Unassembled parts get removed. The overall number of necessary stages and bins are the *stage complexity* and the *bin complexity*. Demaine et al. [3] achieved a number of results that are summarized in Table 1. Most notably, they were able to come up with a system (based on utilizing a spanning tree) that can produce arbitrary shapes in $O(\mathit{diameter})$ many stages, $O(\log n)$ bins and a constant number of glues; the downside is that the resulting shapes are not fully connected.

1.1 Our results

We show that there is a staged assembly system for an arbitrary polyomino with the following properties:

1. polylogarithmic stage complexity
2. constant glue and tile complexity
3. constant scale factor
4. full connectivity

We show how to assemble an arbitrary polyomino even with holes. Most methods of Demaine et al. using a constant number of glues and tiles have either a big stage complexity or the build polyomino is not fully connected. We use a polylogarithmic number of stages with full connectivity and still use a constant number of glues and tiles. On the other hand, we use a small constant scale factor for the assembled shapes. Table 1 gives an overview of the methods which uses a constant number of glues and tiles.

1.2 Related work

As mentioned above, we stick to the aTAM. For other models, see [1]. Other related geometric work by Cannon et al. [2] and Demaine et al. [4] considers reductions between different systems, often based on geometric properties. Fu et al. [6] use geometric tiles in a generalized tile assembly model to assemble shapes [6]. Fekete et al. [5] study the power of using more complicated polyominoes as tiles.

*CSAIL, MIT, USA. edemaine@mit.edu

†Department of Computer Science, TU Braunschweig, Germany. s.fekete@tu-bs.de, arne.schmidt@tu-bs.de

Lines and Squares	**Glues**	**Tiles**	**Bins**	**Stages**	τ	**Scale**	**Conn.**	**Planar**
Line [3] (sect. 2.1)	3	6	7	$O(\log n)$	1	1	full	yes
Square — Jigsaw techn. [3] (sect. 2.2)	9	$O(1)$	$O(1)$	$O(\log n)$	1	1	full	yes
Square — $\tau = 2$ (sect. 2.3)	4	$O(1)$	$O(1)$	$O(\log n)$	2	1	full	yes

Arbitrary Shapes	**Glues**	**Tiles**	**Bins**	**Stages**	τ	**Scale**	**Conn.**	**Planar**
Spanning Tree Method [3]	2	16	$O(\log n)$	$O(diameter)$	1	1	partial	no
Simple Shapes [3]	8	$O(1)$	$O(n)$	$O(n)$	1	2	full	no
Simple Shapes (sect. 3.2)	18	$O(1)$	$O(\|V\|)$	$O(\log^2 n)$	1	4	full	no
Monotone Shapes [3] (sect. 3.1)	9	$O(1)$	$O(n)$	$O(\log n)$	1	1	full	yes
Shape with holes (sect. 3.3)	7	$O(1)$	$O(\|V\|)$	$O(\log^2 n)$	2	3	full	no

Table 1: Overview of some results from Demaine et al. [3] and us using a constant number of glues and tiles. Also pictured are bin complexity, stage complexity, temperature, scale factor, connectivity and planarity. n is the side length of a smalles bounding square and V are the vertices of the polyomino.

Using stages has received attention in DNA self assembly. Reif [8] uses a step-wise model for parallel computing. Park et al. [7] consider assembly techniques with hierarchies to assemble DNA lattices. Somei et al. [10] use a stepwise assembly of DNA tiles. None of these works considers complexity aspects.

2 Lines and Squares

We start with staged assembly for lines and squares. While the results of Section 2.1 and 2.2 are due to [3], Section 2.3 states a new result.

2.1 $1 \times n$ lines

Theorem 1 *A $1 \times n$-line can be assembled with a $\tau = 1$ staged assembly system using $O(\log n)$ stages, 3 glues, 6 tiles and 7 bins.*

2.2 $n \times n$ squares (divide and conquer)

Making use of a "jigsaw" technique, Demaine et al. [3] were able to come up with an efficient method that is

Theorem 2 *An $n \times n$-square can be assembled with full connectivity in $O(\log n)$ stages with 9 glues, $O(1)$ tiles, $O(1)$ bins, and $\tau = 1$.*

2.3 $n \times n$ squares with $\tau = 2$

For $\tau = 2$ assembly systems, it is possible to come up with more efficient ways of constructing a square. The construction is based on an idea by Rothemund and Winfree (see [9]), which we adapt to staged assembly. Basically, it consists of connecting two lines by a corner tile, before filling up this frame; see Figure 1.

Theorem 3 *There exists a $\tau = 2$ assembly system for a fully connected $n \times n$ square with $O(\log n)$ stages, 4 glues, 14 tiles and 7 bins.*

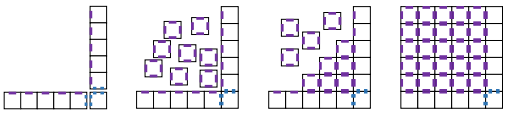


Figure 1: First: Construction for square assembly. Second: Filling up the square. Third: Intermediate Result: some tiles have been assembled to the backbone. Fourth: Fully assembled square.

Proof. First we need to construct the $1 \times (n-1)$ lines with strength 2 glues. We know from Theorem 1 that a line can be constructed in $O(\log n)$ stages, 3 glues, 6 tiles and 7 bins. Because both lines are perpendicular they will not connect. Therefore we can use all 7 bins to construct both lines parallel. For each line we use tiles such that the side, which is directed to the inner side of the square, has a strength 1 glue. In the next stage we mix together the single corner tile with the two lines. In the last step we add a tile with the strength 1 glues on all four sides. When the square is filled no further tile will be connecting because every connection would have a strength sum of 1.

Overall we needed $O(\log n)$ stages with 4 glues (3 for the construction, 1 for filling up the square), 14 tiles (6 for each line, 1 for the corner tile, 1 for filling up the square) and 7 bins for the parallel construction of the two lines. □

3 Assembling Polyominoes

Now we describe approaches for assembling arbitrary polyominoes.

3.1 Monotone shapes

For monotone shapes, [3] showed the following.

Theorem 4 *A monotone polyomino can be assembled with full connectivity in a $\tau = 1$ staged assembly system in $O(\log n)$ stages using 9 glues, $O(1)$ tiles and $O(n)$ bins.*

3.2 Simple shapes

We present a system for building simple polyominoes. The main idea comes from [3], i.e., splitting the polyomino into strips. Then an arbitrary strip gets assembled piece by piece and if there is a component which can assemble to the currently strip then we create the component and attach it to the strip.

Our new staged assembly system partitions the polyomino into rectangles and uses them to assemble the whole polyomino. We may use more bins than the old method, but we have an improvement in the stage complexity. We first consider a building block, see Figure 2. Details are omitted due to limited space.

Lemma 1 *A $2n \times 2n$ square which has at most two tabs each top or left side and at most two pockets each bottom or right side (see Figure 2) can be assembled with $O(\log n)$ stages, 9 glues, $O(1)$ tiles and $O(1)$ bins at $\tau = 1$.*

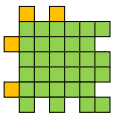


Figure 2: A square (green) with tabs on top and left side (yellow) and pockets on bottom and right side.

This construction works equally well for modified $2n \times 2m$ rectangles. Hence, we can search for a decomposition of a simple polyomino into components of this type.

Theorem 5 *Let V be the set of vertices of the polyomino. There exists a $\tau = 1$ staged assembly system that constructs a fully connected simple polyomino in $O(\log^2 n)$ stages with 18 glues, $O(1)$ tiles, $O(|V|)$ bins and scale factor 4.*

Proof. Details are omitted for lack of space. The main idea is to cut the polyomino into rectangles. These rectangles are recursively divided into subsets, and assembled making use of jigsaw techniques.

Overall we have $O(\log n)$ stages to assemble the $O(|V|)$ rectangles with $O(1)$ bins for each rectangle plus $O(\log^2 n)$ stages to assemble the polyomino from the rectangles which are in total $O(\log^2 n)$ stages and $O(|V|)$ bins. For the rectangles we need 9 glues and 9 glues for the remaining assemblage, hence, in total 18 glues and with it $O(1)$ tiles. To uniquely assemble the polyomino we need to scale it by a factor of 4 in total (scaled by 2 two times). That is, we replace each tile by a 4×4 supertile. □

Figure 3: Left: A chosen rectangle (orange) which splits the polyomino into components (green). Middle: Decomposition of splitting rectangle. Right: Decomposition of the components.

3.3 Temperature $\tau = 2$ assembly

The idea for assembling polyominoes with holes at $\tau = 2$ is similar to squares. To this end, we construct a **backbone**. For an arbitrary polyomino scaled by a factor 3, this is a spanning construct formed with the following line types:

1. the lines on the **left** boundary of the polyomino or **right** boundary of a hole,
2. the lines on the **right** boundary of the polyomino or **left** boundary of a hole,
3. the lines on the **upper** boundary of the polyomino or **lower** boundary of a hole,
4. the lines on the **lower** boundary of the polyomino or **upper** boundary of a hole.
5. the lines that connect two components along the first row of an 3×3 supertile.

In order to avoid cycle in this construction, we leave out the lowest leftmost line on the boundary of the polyomino and the highest rightmost line on the boundary of each hole. Moreover, the fifth line type can be found by moving left from a leftmost (and lowest in case of ties) tile of each hole until a line gets hit. Hence, the construction yields a simple shape. An example for a backbone can be found in Figure 4.

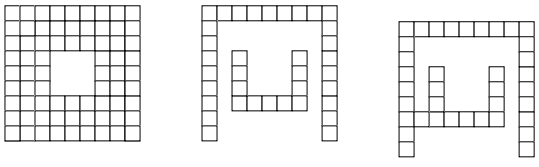


Figure 4: Left: A scaled polyomino with one hole. Middle: Backbone without the fifth type of lines. Right: Complete Backbone of the shape.

Lemma 2 *Let V be the set of vertices of a polyomino P. After scaling P by a factor of 3, the corresponding backbone can be assembled in $O(\log^2 n)$ stages with 4 glues, $O(1)$ tiles and $O(|V|)$ bins.*

Proof. The backbone consists of two types: lines and connection tiles (see Figure 5). Observe that each connection tile has either two or three adjacent lines. Hence, in the adjacency graph they would have degree two and three respectively.

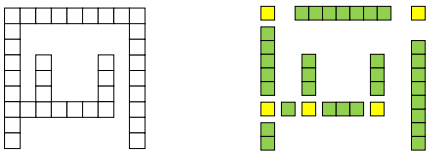


Figure 5: Left: A backbone of a polyomino. Right: A backbone decomposed into lines (green) and connection tiles (yellow).

The construction proceeds in three steps. Details are omitted due to space constraints. In total we use 4 glues, $O(1)$ tiles and $O(|V|)$ bins to assemble a backbone for a given polyomino within $O(\log^2 n + 2 \cdot \log n) = O(\log^2 n)$ stages. □

We can now use this idea to construct any polyomino by assembling its backbone and then filling up:

Theorem 6 *Let V be the set of vertices of the polyomino. Then there is a $\tau = 2$ staged assembly system that constructs a fully connected arbitrary polyomino in $O(\log^2 n)$ stages, 7 glues, $O(1)$ tiles, $O(|V|)$ bins and scale factor 3.*

Proof. The construction proceeds in two parts; again, we have to omit details. For the first part we present a construction such that no tile gets to an unwanted position while filling up the polyomino.

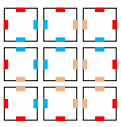


Figure 6: Glue chart for 3×3 tiles to fill up the shape. Blue glue $\triangleq g1$, orange glue $\triangleq g2$ and red glue $\triangleq g3$.

To fill up the polyomino we mix the nine kinds of tiles (see Figure 6) plus the backbone in one bin. In total we need $O(\log^2 n)$ stages, 7 glues, $O(1)$ tiles and $O(|V|)$ bins to assemble a fully connected polyomino, scaled by a factor 3 from the target shape. □

4 Future Work

Our new methods have the same stage and bin complexity and use quite a small number of glues. Because the bin complexity is in $O(|V|)$ for some polyomino with V as the set of vertices, we may need many bins if the polyomino has many vertices. Hence, both methods are good for slightly branched polyominoes. This raises the question for a staged assembly system with the same complexities but a better bin complexity for polyominoes with many vertices, i.e., if $|V| \in O(n^2)$?

Another interesting challenge is to come up with a more efficient system for an arbitrary polyomino. Is there a staged assembly system of stage complexity $o(\log^2 n)$ without increasing the other complexities? In total we think that both methods are a good approach to assemble a polyomino although the number of bins may be a really big value.

References

[1] G. Aggarwal, Q. Cheng, M. H. Goldwasser, M.-Y. Kao, P. M. de Espanes, and R. T. Schweller. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34(6):1493–1515, 2005.

[2] S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. T. Schweller, S. M. Summers, and A. Winslow. Two hands are better than one (up to constant factors). In *STACS*, pages 172–184, 2013.

[3] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Ishaque, E. Rafalin, R. T. Schweller, and D. L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with o(1) glues. *Natural Computing*, 7(3):347–370, 2008.

[4] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow, and D. Woods. One tile to rule them all: Simulating any tile assembly system with a single universal tile. In *ICALP*, pages 368–379, 2014.

[5] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *SODA*, pages 148–167, 2015.

[6] B. Fu, M. J. Patitz, R. T. Schweller, and R. Sheline. Self-assembly with geometric tiles. In *ICALP*, pages 714–725. 2012.

[7] S. H. Park, C. Pistol, S. J. Ahn, J. H. Reif, A. R. Lebeck, C. Dwyer, and T. H. LaBean. Finite-size, fully addressable DNA tile lattices formed by hierarchical assembly procedures. *Angewandte Chemie*, 118(5):749–753, 2006.

[8] J. H. Reif. Local parallel biomolecular computation. *DNA-Based Computers*, 3:217–254, 1999.

[9] P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *STOC*, pages 459–468, 2000.

[10] K. Somei, S. Kaneda, T. Fujii, and S. Murata. A microfluidic device for DNA tile self-assembly. In *DNA Computing*, pages 325–335. Springer, 2006.

[11] H. Wang. Proving theorems by pattern recognition—ii. *Bell system technical journal*, 40(1):1–41, 1961.

[12] E. Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

Caging Polygons by a Finger and a Wall

Bahareh Banyassady* Mansoor Davoodi† Ali Mohades*

Abstract

This paper addresses the problem of caging an arbitrary polygon in the plane by a finger-wall caging grasp, which consists of a finger of a robot arm and a wall. An object is caged by a finger-wall grasp, when it is impossible for the object to move to an arbitrary placement far from its initial placement without penetrating the finger or the wall. We present an algorithm in $O(n^2 \log n)$ time for computing all configurations of the finger-wall grasp which cage a given polygon with n edges. In addition, the output set of all caging grasps can be queried in $O(\log n)$ time to check whether a given arbitrary finger-wall grasp cages the polygon.

1 Introduction

Robotic manipulators are designed to perform a wide variety of tasks in production lines of diverse industrial sectors, such as assembly or part orienting. To perform these tasks the robot arm has to first grasp the object in a proper way. Human being, whom robotic researchers have tried to mimic in many areas, to perform pick and place and also transportation, often grasp the object in a way that it can move among the grasping fingers, but cannot escape through them [1]. The caging problem (or capturing problem) was posed by Kuperberg [2] as a problem of finding the set of all placements of fingers which prevent an object from moving arbitrarily far from its given position.

It is generally assumed in the literature that objects are grasped at some point contacts and idealizations such as a line or surface contact can be approximated by two or more point contacts [3]. Fingertip grasps (point contacts) enable precise control of the object and adroit object manipulation, but limit the amount of force which can be exerted on the work piece. Inner-Link grasps where contacts occur at more than one point along a link are more stable in the face of environmental disturbances and can exert higher forces on a grasped object. However, they are geometrically more complicated and consequently more difficult to analyze [4]. Hence, it is important to look for the other possible practical grasps in order to reduce the number of point contacts. In everyday life, we frequently lean an object against a flat surface, such as a table or a wall, to constrain its motions. In the planar world, the analog of a wall is a supporting line [5]. Thus, we consider finger-wall grasps in which the finger is represented by a point, and the wall is represented by a line.

There are also some researches on caging polygons by three-finger and two-finger grasps. Pipattanasomporn and Sudsang [6], and Vahedi and van der Stappen [7] independently have solved the problem for two-finger grasps in $O(n^2 \log n)$ time, and also constructed a data structure capable of answering queries in $O(\log n)$ time. The running time of both solutions is independent of the complexity of the reported caging grasps. Whereas the number of elements in finger-wall caging grasps are equal to the number of elements in two-finger caging grasps, there are some similarities between these problems solving methods.

In this paper we present an algorithm to find the set of all finger-wall caging grasps of a given polygon in $O(nm \log n)$ where m is the size of the polygon's convex hull. In section 2 some basic definitions and notations are introduced. In section 3 the space configuration is partitioned in order to reduce the search space and in section 4 a graph which represents the search space is constructed. Finally, in section 5 we conclude the paper.

2 Preliminaries and Notations

This section addresses the problem of caging a polygon P by a point and a line. Formally, P is caged by a point and a line when its placement lies in a compact valid region of its free configuration space regarding the point and the line as obstacles. Informally, polygon P is caged by a point and a line when it is impossible to take the object to arbitrary placement far from its initial placement without penetrating the point and/or the line. A placement of the point and the line is a caging configuration if the object is caged by that placement of the point and the line. In general it is easier for the explanation to consider the polygon fixed and to move the point and the line instead while keeping their mutual distances fixed. Therefore, P is caged when it is impossible to rigidly move (translate and/or rotate) the point and the line to infinity without penetrating P.

*Department of Mathematics and Computer Science, Amirkabir University of Technology, Iran. {bahareh.banyasady, mohades}@aut.ac.ir

†Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences, Iran. mdmonfared@iasbs.ac.ir

The given simple polygon P in the plane is bounded by n edges and is assumed to lie inside ω_1 (which ω_i is a circle of radius i centered at O). Finding the set of all valid configurations of the point and the line that cage P is the target in this paper. The bounded configuration space $\mathcal{F} \subset \mathbb{R}^4$ represents the set of all possible configurations of the point and the line. Formally $\mathcal{F} = \mathcal{F}_p \times \mathcal{F}_l$ in which $\mathcal{F}_p$ is the set of all points which lie outside the polygon and inside the ω_1 and $\mathcal{F}_l$ is the set of points which are obtained from a map of all lines which do not intersect the polygon and do intersect the ω_3. It can be proven that any point which lies outside the ω_1 is not the point of any caging configuration, also any line which does not intersect the ω_3 is not the line of any caging configuration.

A unit trajectory of a point is a continuous function $Tp : [0,1] \to \mathcal{F}_p$ that starts at $Tp(0) \in \mathcal{F}_p$ and terminates at $Tp(1) \in \mathcal{F}_p$, where $Tp(t)$ denotes the position of the point on the plane at a normalized time $t \in [0,1]$. A unit trajectory of a line is a continuous function $Tl : [0,1] \to \mathcal{F}_l$ that starts at $Tl(0) \in \mathcal{F}_l$ and terminates at $Tl(1) \in \mathcal{F}_l$, where $Tl(t)$ denotes the position of the line on the plane at a normalized time $t \in [0,1]$. A synchronized trajectory pair (Tp, Tl) represents the movement of the point and the line, such that $(Tp(t), Tl(t))$ denotes the system's configuration at a normalized time $t \in [0,1]$.

For $x = (p,l) \in \mathcal{F}$, we refer to the segment which starts at p and ends at its perpendicular intersection point with l (and does not contain the intersection point) as $h[x)$ or $h[p,l)$. x is a free configuration if $h[x)$ does not intersect the interior of the polygon. A synchronized trajectory pair which starts at $x \in \mathcal{F}$ and terminates at a free configuration is an escape trajectory of x. The Euclidean distance from p to l is represented by $d(x)$. Separation distance of the synchronized trajectory pair (Tp, Tl) is the maximum distance from $Tp(t)$ to $Tl(t)$ during the trajectory, and is represented by $Sd(Tp, Tl) = \max_{0 \le t \le 1} d(Tp(t), Tl(t))$. The critical distance of x is the minimum separation distance of all escape trajectories of x and is represented by $cd(x) = \min_{\forall (Tp,Tl)} Sd(Tp, Tl)$ where (Tp, Tl) is an escape trajectory of x. Obviously, for any $x \in \mathcal{F}$ there is $cd(x) \ge d(x)$.

Proposition 1 *$x \in \mathcal{F}$ is a caging configuration if and only if $cd(x) > d(x)$.*

Caging configurations x and x' are connected if there is a synchronized trajectory pair from x to x' in which all configurations are caging. We refer to a maximal connected set of caging configurations as a maximal caging set.

Lemma 2 *If x is a caging configuration and there is a synchronized trajectory pair (Tp, Tl) from x to x' that $Sd(Tp, Tl) < cd(x)$ then x and x' are connected.*

Lemma 3 *If x and x' are connected, $cd(x) = cd(x')$.*

Regarding the proposition 1 computing the critical distance of all configurations is sufficient to find the caging configurations. According to Lemma 3 the critical distance of all members of a maximal caging set are equal. Hence, in the following, the critical distance of maximal caging sets are computed by partitioning the configuration space and creating a corresponding graph.

3 Configutation Space Partitioning

The vertices of the polygon P in counterclockwise order are called $v_1, v_2, \ldots, v_n$ and its edges are called $e_i = v_i v_{i+1}$ ($1 \le i \le n$ and $v_{n+1} = v_1$). The vertices of the convex hull of P in counterclockwise order are called $V_1, V_2, \ldots, V_m$ in which $m = |CH(P)|$ and its edges are called $E_j = V_j V_{j+1}$ ($1 \le j \le m$ and $V_{m+1} = V_1$). We represent a partitioning for the configuration space, such that the resulting cells of the partitioning do not have intersections with each other unless in their boundaries. $x = (p,l) \in \mathcal{F}$ is in cell $F_{ij} \subset \mathcal{F}$, if e_i is the first edge of P that intersects with segment $h[p,l)$ and V_j is the nearest vertex of $CH(P)$ to line l. And $x = (p,l) \in \mathcal{F}$ is in cell $F_{free} \subset \mathcal{F}$ if $h[p,l)$ does not intersect P. Furthermore, we split some cells into two or three cells in order to make cells reaching some properties. In the following, the two types of split functions are introduced.

For $F_{ij} \subset \mathcal{F}$ we call the line which is along E_{j-1} as the left boundary line or Lb for short, and the line which is along E_j as the right boundary line or Rb for short. Directions of Lb and Rb show the range of directions for the line of F_{ij}'s configurations.

Type 1: If there is a configuration $(p_0, l_0) \in F_{ij}$ in which l_0 passes through V_j and directions of l_0 and e_i are equal, then split the F_{ij} into two cells of F_{ij}^l and F_{ij}^r. Cell F_{ij}^l contains $(p,l) \in F_{ij}$ in which the direction of l is between directions of Lb and l_0, and cell F_{ij}^r contains $(p,l) \in F_{ij}$ in which the direction of l is between directions of l_0 and Rb. By this split the line direction of a cell's configurations are all greater or all smaller than the direction of e_i.

Type 2: For cell F_{ij}^k ($k \in \{null, r, l\}$) and its boundary lines Lb and Rb, we call the endpoint of e_i, which has the smaller summation of distances from Lb and Rb, as nearest endpoint or N for short. If there is a configuration $(p_0, l_0) \in F_{ij}^k$ in which l_0 passes through V_j and is perpendicular to NV_j, then split the F_{ij}^k into two cells of $F_{ij}^{l'}$ and $F_{ij}^{r'}$. Cell $F_{ij}^{l'}$ contains $(p,l) \in F_{ij}^k$ in which the direction of l is between directions of Lb and l_0, and cell $F_{ij}^{r'}$ contains $(p,l) \in F_{ij}^k$ in which the direction of l is between directions of l_0 and Rb. By this split the line direction of a cell's configurations are all greater or all smaller than the direction of the perpendicular line to NV_j.

Finally the number of obtained cells is of the order of $O(nm)$. Regarding the properties of cells it can be observed that, for $(p_0, l_0) \in F_{ij}^k$ ($k \in \{null, r, l, r', l'\}$) in which $p_0 \in e_i$, if p_0 goes to N along e_i while the line is static, the distance of the point and the line decreases in the trajectory. Also for $(N, l_0) \in F_{ij}^k$ in which l_0 passes through V_j, if l_0 rotates around V_j to Lb or Rb while the point is static at N, the distance of the point and the line change strictly monotonic in any of both trajectories. Configuration $x_{min} \in F_{ij}^k$ is a local minima of F_{ij}^k if for all $x \in F_{ij}^k$ there is $d(x_{min}) \leq d(x)$. Therefore, $x_{min} = (N, B)$ where B is either Lb or Rb which is the nearest one to N.

Lemma 4 *If x_{min} is the local minima of F_{ij}^k, for any $x \in F_{ij}^k$ there is a trajectory that starts at x and ends at x_{min} in which the distance of the point and the line decreases.*

Proof. Construct the trajectory which starts at $x = (p, l)$ and continues by getting p and l close to each other along $h[x)$, and terminates at $x' = (p', l')$ in which p' is the intersection of $h[x)$ with e_i and l' is the parallel line to l passing through V_j. The trajectory continues by moving p' along e_i to N while the line l' is static, and then continues by rotating l' to B while the point is static at N. Obviously the final configuration is $x_{min} = (N, B)$ and the distance between the point and the line decreases during the trajectory. □

Lemma 5 *Any cell which is obtained from the configuration space partitioning has intersection with at most one maximal caging set.*

Proof. If x_{min} is the local minima of F_{ij}^k and $x_0 \in F_{ij}^k$ is a caging configuration, there is a trajectory with separation distance of $d(x_0)$ which starts at x_0 and terminates at x_{min} (Lemma 4); thus, x_0 and x_{min} are connected (Lemma 2). In results x_{min}, x and similarly all the other caging configurations of F_{ij}^k are members of the same maximal caging set. □

Conclusion: F_{ij}^k has intersection with a maximal caging set if and only if the local minima of F_{ij}^k is a caging configuration.

Lemma 6 *If x_{min} is the local minima of F_{ij}^k then $\min_{x \in F_{ij}^k} cd(x) = cd(x_{min})$.*

Proof. If x_{min} is not a caging configuration then there is not any caging configuration in F_{ij}^k (Lemma 5) and $cd(x) = d(x)$ for all $x \in F_{ij}^k$. In results $\min cd(x) = \min d(x) = d(x_{min}) = cd(x_{min})$. If x_{min} is a caging configuration, assume to the contrary that there is $x_0 \in F_{ij}^k$ for which $cd(x_0) < cd(x_{min})$. According to Lemma 4 there is a trajectory from x_0 to x_{min} with separation distance of $d(x_0)$ which is $d(x_0) \leq cd(x_0) < cd(x_{min})$. So on the reverse trajectory is the one from the caging configuration x_{min} to x with separation distance of less than $cd(x_{min})$. So $cd(x) = cd(x_{min})$ (Lemma 2), which contradicts our assumption. Thus, $cd(x) \geq cd(x_{min})$ for all $x \in F_{ij}^k$ and it results $\min cd(x) = cd(x_{min})$. □

Theorem 7 *Any $x \in F_{ij}^k$ is a caging configuration if and only if $d(x) < cd(x_{min})$.*

Proof. If x is a caging configuration we have $d(x) < cd(x)$, also x and x_{min} are connected (Lemma 2); and $cd(x) = cd(x_{min})$ (Lemma 3); so, $d(x) < cd(x_{min})$. If $d(x) < cd(x_{min})$ it is equal to $d(x) < \min_{x \in F_{ij}^k} cd(x)$ (Lemma 6) ; hence, $d(x) < cd(x)$ and it means that x is a caging configuration. □

Regarding the Theorem 7, in order to find the set of caging configurations of a cell, it is sufficient to compute the critical distance of the cell's local minima -called critical distance of the cell. In the following, computing the critical distance of cells is discussed.

4 Finding Critical Distance of Cells

Considering $x_b = (p_b, l_b) \in \mathcal{F}$ such that p_b is on the boundary of the polygon and l_b is a tangent line to the polygon. For escape trajectory (Tp, Tl) of x_b, the corresponding squeezed escape trajectory (Tp_s, Tl_s) with smaller separation distance is constructed in 3 steps. In the first step, for $t \in [0, t']$ when t' is the smallest value that $(Tp(t'), Tl(t'))$ is a free configuration, call the first intersection point of $h[Tp(t), Tl(t))$ with the polygon as $Tp'(t)$, and from the two lines which are parallel to $Tl(t)$ and tangent to the polygon call the nearest one to $Tl(t)$ as $Tl'(t)$. Obviously the separation distance of (Tp', Tl') is smaller than or equal to the separation distance of (Tp, Tl). In the second step, if Tp' is discontinuous, add segments between each two consecutive discontinuous points to make the continuous trajectory Tp'', so the separation distance of (Tp'', Tl') and (Tp', Tl') in $[0, t']$ are equal. In the third step, scale t' in order to make a synchronized trajectory pair in $[0, 1]$. We refer to the obtained trajectory as squeezed escape trajectory and represent it with (Tp_s, Tl_s). Obviously $Sd(Tp_s, Tl_s) \leq Sd(Tp, Tl)$. Thus, to find the critical distance of x_b computing the separation distance of all different squeezed escape trajectories is sufficient. Since the local minima of a cell, which was represented by $x_{min} = (N, B)$, has the properties of x_b, in order to find the critical distance of a cell, we will compute the separation distance of all squeezed escape trajectories of the cell's local minima. In the following, the relationship among critical distances of adjacent cells have been formalized.

Consider $B(F_i, F_j)$ represents the set of configurations which are in common boundaries of F_i and F_j. If $x_1 \in B(F_i, F_j)$ and $x_2 \in B(F_j, F_k)$ there are two trajectories, one from x_1 and the other from x_2 to local minima of F_j with separation distances of respectively $d(x_1)$ and $d(x_2)$. Hence, there is a trajectory from x_1 to x_2 with separation distance of $\max\{d(x_1), d(x_2)\}$. It concludes that for a squeezed escape trajectory which starts at x_0 and passes through $F_1, F_2, \ldots, F_f, F_{free}$ the separation distance is $\max_{1 \le i \le f} \{d(x_i)\}$ where $x_i \in B(F_i, F_{i+1})$. Thus, among all squeezed escape trajectories passing through $F_1, \ldots, F_f, F_{free}$ the one which passes through the local minima of common boundaries' configurations has the minimum separation distance.

There is a transition between each two cells F_i and F_j if $B(F_i, F_j) \neq \emptyset$, and the cost of the transition is equal to the minimum of $d(x)$ for all $x \in B(F_i, F_j)$. In the worst case situation, the number of transitions that a cell is associated with can be as high as $O(n)$. Fortunately, we found out there are only $O(1)$ transitions associating with a cell, which are called basic transitions, that have effects on the computation of separation distance of any squeezed escape trajectory.

Theorem 8 *If there is a transition between F_1 and F_2 with the cost of $d(x_{12})$, there is a sequence of basic transitions that starts at F_1 and ends at F_2, in which the maximum cost of the basic transitions are less than or equal to $d(x_{12})$.*

For $x_{min} \in F_i$, any squeezed escape trajectory is corresponding to a sequence of cells starting at F_i and ending at F_{free}. Thus in order to compute the $cd(x_{min})$ which is the minimum separation distance of all squeezed escape trajectories for x_{min}, we should find the minimum of maximum of basic transitions' cost in any sequence of cells starting at F_i and ending at F_{free}. In this paper the basic transitions are not described; however, finding the basic transitions and their cost is of order $O(nm \log n)$ by running $O(nm)$ times of Ray shooting algorithm. From insights we have obtained so far, we are now ready to concrete the definition of our search space.

Connectivity graph $G = (V_G, E_G)$ is a weighted graph that any vertex $vg_i \in V_G$ corresponds to cell $F_i \subset \mathcal{F}$ and any edge $e_{ij} \in E_G$ corresponds to a basic transition between two cells F_i and F_j, and its weight is equal to the cost of the basic transition which is called w_{ij}. Also critical distance of $vg_i \in V_G$ is equal to the critical distance of F_i which is called $D(vg_i)$.

Theorem 9 *For any $vg_i \in V_G$ if $N(vg_i) \subset V_G$ is the set of all vertices which are adjacent to vg_i, then* $D(vg_i) = \min_{vg_j \in N(vg_i)} (\max(D(vg_j), w_{ij}))$.

According to the Theorem 9 if vg_i and vg_j are adjacent vertices of G then $D(vg_i) \le \max\{(D(vg_i), w_{ij})\}$; To compute the critical distances of vertices we present a slightly modified Dijkstra algorithm. The first step of the algorithm is to initialize the critical distance of all vertices as infinite except the critical distance of vg_{free}, which corresponds to cell $F_{free} \subset \mathcal{F}$, as zero. In each step a vertex vg_i, which has the minimum critical distance between all the not selected vertices, is selected and the critical distance of its adjacent vertices are updated; If the present critical distance of $vg_j \in N(vg_i)$ is greater than $\max\{(D(vg_i), w_{ij})\}$ set $\max\{(D(vg_i), w_{ij})\}$ as the updated critical distance of vg_j. Then add the vg_i to the set of selected vertices. Do the same procedure until all the vertices have been selected. Finally, by having critical distance of vertices, the set of all caging configurations of any cell F_i, which is represented by $C_i = \{x \in F_i | d(x) < D(F_i) = D(vg_i)\}$, can be computed. To decide whether a given configuration x is a caging configuration or not, find cell F_i that $x \in F_i$ by running the Ray shooting algorithm in $O(\log n)$, then $d(x)$ and $D(F_i)$ should be compared.

5 Conclusion

It was discussed that computing the critical distance ofl maximal caging sets is sufficient to find all the caging configurations. Thus, the configuration space was partitioned and the critical distance of cells was computed by searching in a graph. Computing the vertices and the edges of the connectivity graph are respectively of the order of $O(nm)$ and $O(nm \log n)$, searching the graph to compute the critical distance of cells is also $O(nm \log n)$. Therefore, the running time of the algorithm is $O(nm \log n)$ and the query can be answered in $O(\log n)$ time.

References

[1] M. Vahedi. *Caging Polygons with Two and Three Fingers.* Phd thesis, Department of Information and Computing Science, Utrecht University, 2009.

[2] W. Kuperberg. *Problems on Polytopes and convex sets.* DIMACS Workshop on Polytopes, 1990.

[3] A. Bicchi, V. Kumar. *Robotic Grasping and Contact: A Review.* In ICRA, IEEE, 2000.

[4] B. S. Eberman. *Whole Arm Manipulation: Kinematics and Control.* Thesis, Department of Mechanical Engineering, Massachusetts Institute of Tech, 1989.

[5] C. Wentink, A. F. van der Stappen, Mark Overmars. *Algorithms for Fixture Design.* UU-CS, 1996.

[6] P. Pipattanasomporn, A. Sudsang. *Two-finger caging of nonconvex polytopes.* IEEE Transaction on Robotics, 2011.

[7] M. Vahedi, A. F. van der Stappen. *Towards Output-Sensitive Computation of Two-Finger Caging Grasps.* In CASE, 2008.

Subquadratic Medial-Axis Approximation for smooth Curves in $\mathbb{R}^3$

Christian Scheffer*
Department of Computer Science, TU Braunschweig
scheffer@ibr.cs.tu-bs.de

Abstract

We present the first algorithm to approximate the medial axis of a smooth, closed curve $\gamma \subset \mathbb{R}^3$ in subquadratic time. Our algorithm works on a sufficient dense ε-sample and comes with a convergence guaranty for the non-discrete but continuous approximation object.

1 Introduction

The *medial axis* of a geometric shape was originally introduced by Blum [4] and is defined as follows:

Definition 1 *The medial axis $M_\gamma \subset \mathbb{R}^3$ of a smooth, closed curve $\gamma \subset \mathbb{R}^3$ is defined as the closure of all points which have at least two closest points from γ.*

Using the medial axis as an implicit representation of the underlying (smooth) shape γ, the concept of the medial axis has a large variety of applications in theory and practice, see e.g. Dey and Zhao as a survey [8]. Often, γ is just given by a discrete point sample. To analyze the density of such point samples, Amenta et al. [2] introduced the *local feature size* $lfs(\cdot)$. It is defined as the function which maps each point from γ onto its distance to M_γ. Based on this, Amenta et al. defined an *ε-sample* S as a point sample on γ so that for each $x \in \gamma$ there is at least one sample point $s \in S$ with $|xs| \leqslant \varepsilon lfs(x)$.

In this paper, we aim for an approach which computes a continuous approximation object $\widetilde{M}_\gamma$ of M_γ so that $\widetilde{M}_\gamma$ converges M_γ for $\varepsilon \to 0$, only based on an ε-sample and in subquadratic time.

Similar to the surface case, the medial axis of a curve may be a highly unstable object so that arbitrary small changes of γ imply large modifications of the medial axis—see [12] for an illustration in the surface case. Hence, the aim to compute an object which realizes a Hausdorff distance to the medial axis which is related to the sampling density, seems to be too ambitious. An alternative approximation quality would be a convergence guarantee for the computed approximation object, as has already been done in previous approaches which deal with similar problem settings [2, 3, 5, 8, 12].

*This research was supported by DAAD grant 57052155.

To our best knowledge, our approach is the first one approximating the medial axis M_γ of an underlying and unknown smooth curve $\gamma \subset \mathbb{R}^3$ by a piecewise-linear object, while it simultaneously guarantees a convergence guarantee and a sub quadratic running time w.r.t. $n := |S|$.

2 Preliminaries

Previous algorithms from literature which approximate the medial axes of curves, compute the Voronoi diagram as a subroutine which in turn leads to a quadratic running time in $\mathbb{R}^3$ [2, 3, 5].

In [12], Scheffer and Vahrenhold present an algorithm to approximate the medial axis M_Γ of an underlying surface Γ, while providing the same guarantees as we aim for, i.e., a convergence guarantee, a subquadratic running time, and a continuous approximation object. From a high-level point of view, their approach is the following: First compute an asymptotical lower bound $\widetilde{\varepsilon}$ for the sampling density ε. Then approximate for each sample point s its Voronoi cell $Vor(s)$ up to an error related to $\widetilde{\varepsilon}$ and hence related to the sampling density ε. After that, read out objects from the approximation of $Vor(s)$ and triangulate these objects by applying a reconstruction of Γ.

To approximate M_Γ, Scheffer and Vahrenhold [12] compute for each sample point s the two *approximate pole points* which are defined as the two farthest vertices of the piecewise-linear approximation of $Vor(s)$ and which lie on different sides of M_Γ. A key property of an approximate pole point p_s of a sample point s is that it is an appropriate approximation of the normal n_s to the surface in s, i.e., $\angle(sp_s, n_s) \in \mathcal{O}(\varepsilon)$.

In our case, the normals to the curve γ in a sample point s are not given via a line but by a plane which is pierced by γ in s. Hence, for approximating the whole medial axis M_γ of a $\gamma \in \mathbb{R}^3$ it is not sufficient to compute only two approximate pole points for each sample point.

Finally, Scheffer and Vahrenhold triangulate the approximate pole points by just mapping the topology of a surface reconstruction from the sample points to the approximate pole points. This works since for each sample point there is one approximate pole point in the interior of the surface and one approximate pole point in the outside of the surface. In our case, we

have for each sample point a set of points, see below, which has to be triangulated so that a one-to-one mapping would not be productive for a continuous approximation object.

3 The Algorithm

To apply the global approach from [12], there are two issues needed to be addressed:

(1) In our case, the construction of two approximate pole points do not provide an appropriate approximation of the normal plane to γ in a sample point s. Thus, we compute for each sample point s a point set $\mathcal{E}_s$, which we call the *equator of* s and which shapes a ring around γ in s.

To obtain a convergence guarantee for the equators and hence for our whole approximation object, we have to relate the density of the equators and the quality of the approximation of the Voronoi cells to the sampling density ε. Unfortunately, the value of ε is not computable which is caused by the instability of the medial axis. But what we can do is, similar as in [12], that we can compute an asymptotical lower bound $\widetilde{\varepsilon}$ for the sampling density—see Algorithm 4. By relating the density of the equators and the quality of the approximation of the Voronoi cells to $\widetilde{\varepsilon}$, we obtain the desired convergence guarantee for the equators.

(2) In our case, γ is a one-dimensional shape, while the union of the equators induces a two-dimensional structure. This implies that only a reconstruction of γ is not able to cover the topology of the equators. Hence, we combine the topology of γ with the order of the equators around γ. This leads to a two-dimensional structure. In particular, we triangulate the equator points of each pair of sample points which lie neighboured on γ seperately. Hence, we first have to compute a piecewise-linear reconstruction N which is homeomorphic to γ—see Algorithm 2.

3.1 Outline of the Algorithm

First, we compute a homeomorphic curve reconstruction N of γ in near-linear time. Second, we compute an asymptotical lower bound $\widetilde{\varepsilon}$ for the sampling density by combining a preliminary version of the equators of constant density with a so called *control function* ψ. After that, we compute the final version of the equators, whose densities are related to $\widetilde{\varepsilon}$. And finally, we triangulate the equators by combining N with the ordering of the equators around N.

The resulting overall approach is formulated by Algorithm 1.

3.2 Near-linear time Curve Reconstruction

Dey and Kumar [7] proposed (by the NN-CRUST) an approach which reconstructs a smooth closed 1-manifold only based on an ε-sample $S \subset \gamma$ in any dimension. As a subroutine, they compute the Voronoi diagram $VD(S)$ of the entire point cloud S. Unfortunately, this implies a worst-case running time of $\mathcal{O}(n^2)$ in $\mathbb{R}^3$. To avoid this in the context of curve reconstruction in $\mathbb{R}^3$, we apply the approach of *approximate neighborhoods*.

Algorithm 1 Approximating M_Γ by a simplicial complex with a stable convergence guarantee in sub quadratic time

1: **function** ApproxMA(S, κ)
2: $N := (S, E) :=$ AN-CRUST(S)
3: $\delta \leftarrow \frac{1}{4000}$
4: $\mathcal{E} :=$ Equators(S, N, δ)
5: **for all** $s \in S$ **do**
6: $\psi(s) \rightarrow \max\left\{\frac{|s_s^- s|}{2}, \frac{|s_s^+ s|}{2}\right\}$
7: $\widetilde{\varepsilon} :=$ ApproxEpsilon($S, \mathcal{E}, \psi$)
8: $\delta := \widetilde{\varepsilon}^{\frac{1}{\kappa}}$
9: $\mathcal{E} :=$ Equators(S, N, δ)
10: Initialize $\widetilde{M}_\gamma$ to be the empty set.
11: $\widetilde{M}_\gamma \leftarrow$ TriangEquatorPoints($S, \mathcal{E}, E$)
12: **return** $\widetilde{M}_\gamma$

Definition 2 ([11, 12]) *Let $X \subset \mathbb{R}^3$ be a discrete point set and let x be an arbitrary point in X. An approximate neighborhood $AN_{\sigma_N}(x)$ of x w.r.t. X is defined as a subset of X, such that there exists a set of cones $\mathcal{C}_{\sigma_N}(s)$, with apex at x and an angular radius of $\sigma_N \geqslant 0$ that covers $\mathbb{R}^3$, such that the following holds: A point $x' \in X$ belongs to the approximate neighborhood $AN_{\sigma_N}(x)$ if, and only if, there is a cone $C \in \mathcal{C}_{\sigma_N}(s)$ such that x' is the point in $C_{\sigma_N}(s) \cap X$ that minimizes the distance from its orthogonal projection onto the axis of C to x.*

In the definition given above, Ruppert and Seidel [11] assume that the point which minimizes the distance of its orthogonal projection, is uniquely defined, e.g., by breaking ties according to an arbitrary but fixed numbering of the points, and allow the axis of a cone C to be an arbitrary ray R starting from the cone's apex x and lying inside the cone. The angular radius of C is defined as $\arg\min_{\sigma_N \geqslant 0}\{\angle(xa, R) \leqslant \sigma_N \mid a \in C\}$.

Lemma 1 ([11]) *For an arbitrary point set $X \subset \mathbb{R}^3$ and $\sigma_N > 0$, the approximate neighborhoods for all points from X can be computed in overall time $\mathcal{O}\left(|X|/\sigma_N^2 \cdot \log^2(|X|)\right)$.*

By applying the approximate neighborhoods of all sample points, we can give an algorithm which reconstructs γ in near-linear time—see Algorithm 2.

Since we choose $\sigma_N \in \mathcal{O}(1)$, we get the desired running time for our AN-CRUST algorithm.

Algorithm 2 Near-linear curve reconstruction

1: **function** AN-CRUST(S)
2: $E \leftarrow \varnothing$
3: $N \leftarrow (S, E)$
4: $\delta \leftarrow \frac{1}{10}$
5: **for all** $s \in S$ **do**
6: Comp. $AN_{\delta/20}(s)$ ▷ Definition 2
7: $s'_s \leftarrow \mathrm{NN}_{AN_{\delta/20}(s)}(s)$
8: $\mathrm{HS}_s \leftarrow \left\{x \in \mathbb{R}^3 | \angle(s'_s s x) \geqslant \frac{\pi}{2}\right\}$
9: $s''_s \leftarrow \mathrm{NN}_{AN_{\delta/20}(s) \cap \mathrm{HS}_s}(s)$
10: $E \leftarrow E \cup \{(s, s'_s), (s, s''_s)\}$
11: **return** N

Corollary 2 *Based on a sufficient dense ε-sample,* AN-CRUST *computes a piecewise-linear reconstruction N of γ which is homeomorphic to γ in $\mathcal{O}\left(n \cdot \log^2(n)\right)$ time.*

3.3 Computing the Equators

For an arbitrarily chosen sample point $s \in S$, we denote the successor and predecessor of s w.r.t. N by s_s^+ and s_s^-. In addition to this, we denote the line in the plane induced by s_s^+, s_s^- and s, with $s \in \widetilde{t}_s$ so that $\angle\left(s_s^- s \widetilde{t}_s\right) = \angle\left(s_s^+ s \widetilde{t}_s\right)$ by $\widetilde{t}_s$. If s_s^-, s and s_s^+ lie collinear, we define $\widetilde{t}_s$ such that $s_s^+, s_s^-, s \in \widetilde{t}_s$ holds. Furthermore, we denote the plan which lies orthogonal to $\widetilde{t}_s$ in s by $\widetilde{N}_s$.

To define the equator of s, we consider a so called *screw* around s. Informally spoken, a *screw* $\mathcal{H}_s$ of s, w.r.t. to an angle σ_H is a partitioning of the Voronoi cell of s w.r.t. the approximate neighborhood of s.

Definition 3 *Let $s \in S$ and $\sigma_H > 0$ be chosen arbitrarily and let $R_1, ..., R_k$, for $k = \lceil (40 \cdot 2 \cdot \pi)/\sigma_H \rceil$ be a set of (counterclockwise) ordered rays starting from s so that $R_i \subset \widetilde{N}_s$ and $\angle(R_i, R_{i+1}) \leqslant \frac{\sigma_H}{40}$ hold. For all $i \in \{1, ..., k\}$, let H_s^i be the halfplane which is bounded by $\widetilde{t}_s$ so that $R_i \subset H_s^i$ holds. We define $\mathcal{H}_s^i$ as the intersection of the Voronoi cell of s, w.r.t. $AN_{\sigma_H/20}(s) \cup \{s_s^+, s_s^-\}$, with the space sandwiched between H_s^i and H_s^{i+1}.*

A screw $\mathcal{H}_s$ of s w.r.t. σ_H is defined as the set of all intersections between two paddles H_s^i and H_s^{i+1} as constructed above.

The equator of s is defined as the union of the farthest points which lie between the paddles of a screw.

Definition 4 *An equator point e_s^i of s, w.r.t. $\mathcal{H}_s^i$ is defined as a farthest point from $\mathcal{H}_s^i$. Based on this, we define the equator $\mathcal{E}_s$ of s as $\bigcup_{i=1,...,k} e_s^i$.*

Definition 4 leads to Algorithm 3 which computes the union $\mathcal{E}$ of all equators, while their density is related to an input parameter δ.

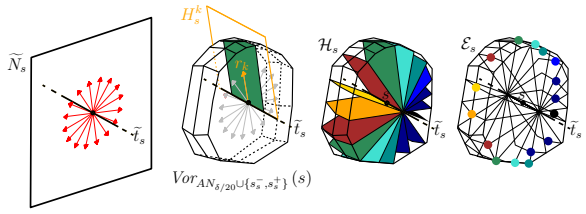


Figure 1: Constructive definition of the equator $\mathcal{E}_s$.

Algorithm 3 Computing the Equators.

function EQUATORS(S, N, δ)
for all $s \in S$ **do**
Comp. $AN_{\delta/20}(s)$ ▷ Definition 2
Comp. $Vor_{AN_{\delta/20}(s) \cup \{s_s^+, s_s^-\}}(s)$ ▷ [10]
Comp. $\widetilde{t}_s$ and $\widetilde{N}_s$
Comp. $\mathcal{H}_s$ w.r.t. δ ▷ Definition 3
Comp. equator $\mathcal{E}_s$ of s ▷ Definition 4
$\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{E}_s\}$
return $\mathcal{E}$

Lemma 3 *Algorithm 3 has a running time of $\mathcal{O}\left(\frac{|S|}{\delta^3} \cdot \max\left\{\log^2(n), \log^2\left(\frac{1}{\delta^2}\right)\right\}\right)$.*

By applying Algorithm 3 twice in a bootstrapping approach, see Steps 4 and 9 of Algorithm 1, we can guarantee that each equator point lies close to the medial axis at which "close" will be related to a function involving exclusively ε. In particular, we first guarantee that the distance to the equators which are computed in the first iteration of Algorithm 3 is an asymptotical upper bound for $lfs(\cdot)$. For this, we can show the following result.

Lemma 4 $dist\left(e_s^i, M_\gamma\right) \leqslant 15 \cdot \sqrt{\delta + \varepsilon} \cdot |se_s^i|$

Since the local feature size is defined as the minimum distance to the medial axis and since $\varepsilon, \delta \in \mathcal{O}(1)$, the distance function to $\mathcal{E}$ can be shown to be an asymptotical upper bound for $lfs(\cdot)$.

Lemma 5 $lfs(s) \leqslant 1.4 \cdot dist(\mathcal{E}, s)$

In the following, we show how to apply Lemma 5 to obtain an asymptotical lower bound for the sampling density which in turn is applied for the computation of the final version of the equators.

3.4 Computing a lower bound $\widetilde{\varepsilon}$ for ε

We compute $\widetilde{\varepsilon}$ as the ratio of a lower bound for $\varepsilon lfs(\cdot)$ and an asymptotical upper bound for $lfs(\cdot)$.

Lemma 5 implies that $dist(\mathcal{E}, s)$ is an asymptotical upper bound for $lfs(\cdot)$. But, since one of our main goals is to achieve a sub quadratic running time, we do not compute $dist(\mathcal{E}, .)$ exactly for all sample points, but approximate it by applying the following result.

Theorem 6 (Chan [6] - Theorem 2.5) *Let $X \subset \mathbb{R}^3$ be a given pointset. For a query point $q \in \mathbb{R}^3$ one can find an $(1+\xi)$-approximate nearest neighbor in $\mathcal{O}\left(\xi^{1-3/2} \cdot \log^2(|X|)\right)$ time with $\mathcal{O}\left(\xi^{1-2/3} \cdot |X| \cdot log(|X|)\right)$ preprocessing time and space.*

We denote the resulting approximate distance function to $\mathcal{E}$ by $\widetilde{lfs}(.)$. Since the application of Chan's algorithm reports equator points within a distance of at least $dist(\mathcal{E}, .)$, Lemma 5 implies $lfs(s) \leqslant 1.4 \cdot \widetilde{lfs}(s)$.

Hence, we finally have to compute an upper bound for $\varepsilon lfs(\cdot)$. We do this by computing a so called *control function* ψ. In particular, we define $\psi(s)$ as $\max\{|s_s^- s|/2, |s_s^+ s|/2\}$. Since S is an ε-sample, we can show that $\psi(s) \leqslant \frac{\varepsilon}{1-\varepsilon} lfs(s)$ holds, for all $s \in S$.

Finally, we define $\widetilde{\varepsilon} := \max_{s \in S}\{\psi(s)/\widetilde{lfs}(s)\}$. The resulting approach is given by Algorithm 4.

Algorithm 4 Computing a non-trivial lower bound for ε

```
1: function ApproxEpsilon(S, ψ, E)
2:     Preprocessing for ANN-queries   ▷ Theorem 6
3:     ξ = 1/10
4:     for all s ∈ S do
5:         lfs~(s) := |s, ANN(s, ξ, E)|   ▷ Theorem 6
6:     ε~ := max_{s∈S} { ψ(s) / lfs~(s) }
       return ε~
```

Corollary 7 *Algorithm 4 runs in $\mathcal{O}\left(n \cdot \log^2(n)\right)$.*

For the second iteration of Algorithm 3, we choose δ related to $\widetilde{\varepsilon}$. Since the value of δ has an influence on the running time of Algorithm 3, we have to guarantee that the inverse of $\widetilde{\varepsilon}$ is upper-bounded by a polynomial of n.

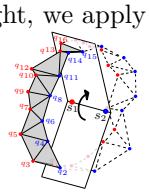
Lemma 8 $\Theta\left(\frac{1}{n}\right) \leqslant \widetilde{\varepsilon} \leqslant 1.5 \cdot \varepsilon$

By combining Lemma 4 and Lemma 8, we can show the following result.

Corollary 9 $\lim_{\varepsilon \to 0}(\mathcal{E}) = M_\gamma$.

3.5 Triangulating $\mathcal{E}$

We obtain an appropriate triangulation of $\mathcal{E}$ by triangulating each pair of equators of two connected sample points separately. In particular, see right, we apply the sweep plane paradigm for each edge $e = (s_1, s_2) \in N$ separately as follows: Let H be an arbitrarily chosen half-plane which is bounded by the line induced by s_1 and s_2. In a nutshell, the desired triangulation is derived by rotating H around e. For each point in time during this rotation process, we store the two last visited equator points q^1 and q^2 from $\mathcal{E}_{s_1}$ and $\mathcal{E}_{s_2}$. Each time when H meets a new equator point q, we add the triangle $\triangle(q^1, q^2, q_j)$, if q^1, q^2 and q_j do not lie at infinity.



Since the triangulating process described above has a near-linear running time w.r.t. the number of equator points, we can upper-bound its running time by $\mathcal{O}\left(n\delta^{-1}\log(\delta^{-1})\right)$. By applying a similar approach to the proof of Corollary 9, we get our main result.

Corollary 10 *Algorithm 1 computes a piecewise-linear approximation $\widetilde{M}_\gamma$ of M_γ in $\mathcal{O}(n^{1+\frac{3}{\kappa}} \cdot \log^2(n))$ time, such that $\lim_{\varepsilon \to 0} \widetilde{M}_\gamma = M_\gamma$ holds.*

References

[1] Aichholzer, O., Aurenhammer, F., Hackl, T., Kornberger, B., Plantinga, S., Rote, G., Sturm, A., Vegter, G.: Recovering Structure from r-Sampled Objects. Computer Graphics Forum **28**(5), 1349–1360 (2009)

[2] Amenta, N., Bern, M.W., Eppstein, D.: The crust and the beta-skeleton: Combinatorial curve reconstruction. Graphical Models and Image Processing **60**(2), 125–135 (1998)

[3] Amenta, N., Choi, S., Kolluri, R.K.: The power crust, union of balls, and the medial axis transform. Computational Geometry: Theory and Applications **19**(2–3), 127–153 (2001)

[4] Blum, H.: A Transformation for Extracting New Descriptors of Shape. In: W. Wathen-Dunn (ed.) Models for the Perception of Speech and Visual Form, pp. 362–380. MIT Press (1967)

[5] Brandt , J. W.: Convergence and Continuity Criteria for Discrete Approximations of the Continuous Planar Skeleton. In: CVGIP Image Understanding **59**(1), pp. 116–124 (2002)

[6] Chan, T.M.: Approximate nearest neighbor queries revisited. Discrete & Computational Geometry **20**(3), 359–373 (1998)

[7] Dey, T.K., Kumar, P.: A simple provable algorithm for curve reconstruction. In: SODA, pp. 893–894 (1999)

[8] Dey, T.K., Zhao, W.: Approximating the medial axis from the Voronoi diagram with a convergence guarantee. Algorithmica **38**(1), 179–200 (2003)

[9] luh Liu, T.: Approximate tree matching and shape similarity. In: In Proceedings, 7th International Conference on Computer Vision, pp. 456–462 (1999)

[10] Preparata, F.P., Shamos, M.I.: Computational Geometry. An Introduction, second edn. Springer, Berlin (1988)

[11] R. Seidel, J.R.: Approximating the d-dimensional complete euclidean graph. In: Proceedings of the Third Annual Conference on Computational Geometry, pp. 207–210 (1991)

[12] C. Scheffer and J. Vahrenhold. Subquadratic medial-axis approximation in $\mathbb{R}^3$. *https://www.dropbox.com/s/42gwget0b5ur1jz/mas.pdf?dl=0*. Technical Report.

Analysis-suitable adaptive T-mesh refinement with linear complexity *

Philipp Morgenstern† Daniel Peterseim‡

Abstract

The following extended abstract summarizes the article [11]. We present an efficient adaptive refinement procedure that preserves analysis-suitability of the T-mesh, this is, the linear independence of the T-spline blending functions. We prove analysis-suitability of the overlays and boundedness of their cardinalities, nestedness of the generated T-spline spaces, and linear computational complexity of the refinement procedure in terms of the number of marked and generated mesh elements.

1 Introduction

T-splines [14] have been introduced as a free-form geometric technology and are one of the most promising features in the Isogeometric Analysis (IGA) framework introduced by Hughes, Cottrell and Basilevs [8, 4].

Since T-splines can be locally refined [13], they potentially link the powerful geometric concept of Non-Uniform Rational B-Splines (NURBS) to meshes with T-junctions (referred as "hanging nodes" in the Finite Element context) and, hence, the well-established framework of adaptive mesh refinement. However, in [1], it was shown that T-meshes can induce linearly dependent T-spline blending functions. This prohibits the use of T-splines as a basis for analytical purposes such as solving a partial differential equation. In particular, the mesh refinement algorithm presented in [13] does not preserve analysis-suitability in general. This insight motivated the research on T-meshes that guarantee the linear independence of the corresponding T-spline blending functions, referred to as *analysis-suitable T-meshes*. Analysis-suitability has been characterized in terms of topological mesh properties in $2d$ [10] and, in an alternative approach, through the equivalent concept of Dual-Compatibility [5], which allows for generalization to three-dimensional meshes.

A refinement procedure that preserves the analysis-suitability of two-dimensional T-meshes was finally presented in [12]. The procedure first refines the marked elements, producing a mesh that is not analysis-suitable in general, and then computes a refinement which is analysis-suitable and generates a T-spline space that is a superspace of the previous one. This second refinement involves heuristic local estimates on how much refinement is needed to achieve the desired properties. Hence, the reliable theoretical analysis of the algorithm is very difficult and so is the analysis of corresponding automatic mesh refinement algorithms driven by a posteriori error estimators. Such analysis is currently available only for triangular meshes [2, 3, 15], but is necessary to reliably point out the advantages of adaptive mesh refinement.

In this paper, we present a new refinement algorithm which provides

1. the preservation of analysis-suitability and nestedness of the generated T-spline spaces,
2. a bounded cardinality of the overlay (which is the coarsest common refinement of two meshes),
3. linear computational complexity of the refinement procedure in the sense that there is a constant bound, depending only on the polynomial degree of the T-spline blending functions, on the ratio between the number of generated elements in the fine mesh and the number of marked elements in all refinement steps.

2 Adaptive mesh refinement

This section defines the new refinement algorithm and characterizes the class of meshes which is generated by this algorithm. The initial mesh is assumed to have a very simple structure. In the context of IGA, the partitioned rectangular domain is referred to as *index domain*. This is, we assume that the *physical domain* (on which, e.g., a PDE is to be solved) is obtained by a continuous map from the active region, which is a subset of the index domain. Throughout this paper, we focus on the mesh refinement only, and therefore we will only consider the index domain. For the parametrization and refinement of the T-spline blending functions, we refer to [12].

*The authors gratefully acknowledge support by the Deutsche Forschungsgemeinschaft in the Priority Program 1748 "Reliable simulation techniques in solid mechanics. Development of non-standard discretization methods, mechanical and mathematical analysis" under the project "Adaptive isogeometric modeling of propagating strong discontinuities in heterogeneous materials".

†Institute for Numerical Simulation, Rheinische Friedrich-Wilhelms-Universität Bonn, Germany

‡Institute for Numerical Simulation, Rheinische Friedrich-Wilhelms-Universität Bonn, Germany

Definition 1 (Initial mesh, element) *Given positive numbers $M, N \in \mathbb{N}$, the initial mesh $\mathcal{G}_0$ is a tensor product mesh consisting of closed squares (also denoted* elements*) with side length 1, i.e.,*

$$\mathcal{G}_0 := \big\{[m-1,m] \times [n-1,n] \mid m \in \{1,\dots,M\}, \; n \in \{1,\dots,N\}\big\}.$$

The domain which is partitioned by $\mathcal{G}_0$ is denoted by $\overline{\Omega} := \bigcup \mathcal{G}_0$.

The key property of the refinement algorithm will be that refinement of an element K is allowed only if elements in a certain neighbourhood are sufficiently fine.

Definition 2 (Level) *The* level *of an element K is defined by*

$$\ell(K) := -\log_2 |K|,$$

where $|K|$ denotes the volume of K. This implies that all elements of the initial mesh have level zero and that the bisection of an element K yields two elements of level $\ell(K)+1$.

Definition 3 (Vector-valued distance) *Given $x \in \overline{\Omega}$ and an element K, we define their distance as the componentwise absolute value of the difference between x and the midpoint of K,*

$$\operatorname{Dist}(K,x) := \operatorname{abs}\big(\operatorname{mid}(K) - x\big) \in \mathbb{R}^2.$$

For two elements K_1, K_2, we define the shorthand notation

$$\operatorname{Dist}(K_1,K_2) := \operatorname{abs}\big(\operatorname{mid}(K_1) - \operatorname{mid}(K_2)\big).$$

Definition 4 *Given an element K and polynomial degrees p and q, the (p,q)-patch is defined by*

$$\mathcal{G}^{p,q}(K) := \big\{K' \in \mathcal{G} \mid \operatorname{Dist}(K',K) \le \mathbf{D}^{p,q}(\ell(K))\big\},$$

where

$$\mathbf{D}^{p,q}(k) = \begin{cases} 2^{-k/2}\left(\lfloor \frac{p}{2} \rfloor + \frac{1}{2}, \lceil \frac{q}{2} \rceil + \frac{1}{2}\right) & \text{if } k \text{ is even,} \\ 2^{-(k+1)/2}\left(\lceil \frac{p}{2} \rceil + \frac{1}{2}, 2\lfloor \frac{q}{2} \rfloor + 1\right) & \text{if } k \text{ is odd.} \end{cases}$$

Definition 5 (Bisection) *Given a mesh $\mathcal{G}$ and an element $K \in \mathcal{G}$, we denote by* $\operatorname{bisect}(\mathcal{G},K)$ *the mesh that results from a level-dependent bisection of K,*

$$\operatorname{bisect}(\mathcal{G},K) := \mathcal{G} \setminus \{K\} \cup \operatorname{child}(K),$$

$$\text{with } \operatorname{child}(K) := \begin{cases} \operatorname{bisect}_{\mathrm{x}}(K) & \text{if } \ell(K) \text{ is even,} \\ \operatorname{bisect}_{\mathrm{y}}(K) & \text{if } \ell(K) \text{ is odd,} \end{cases}$$

where $\operatorname{bisect}_{\mathrm{x}}$ *(resp.* $\operatorname{bisect}_{\mathrm{y}}$*) denotes a bisection in the first (resp. second) dimension.*

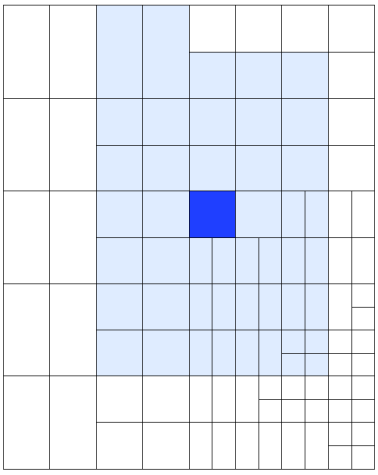


Figure 1: Example of the patch $\mathcal{G}^{p,q}(K)$ in non-uniform mesh for even $\ell(K)$ and $p = q = 5$. K is marked in blue, and $\mathcal{G}^{p,q}(K)$ is highlighted in light blue.

Definition 6 (Multiple bisections) *We introduce the shorthand notation* $\operatorname{bisect}(\mathcal{G},\mathcal{M})$ *for the bisection of several elements $\mathcal{M} = \{K_1,\dots,K_J\} \subseteq \mathcal{G}$, defined by successive bisections in an arbitrary order,*

$$\operatorname{bisect}(\mathcal{G},\mathcal{M}) := \operatorname{bisect}(\dots \operatorname{bisect}(\mathcal{G},K_1)\dots,K_J).$$

Algorithm 1 (Closure) *Given a mesh $\mathcal{G}$ and a set of marked elements $\mathcal{M} \subseteq \mathcal{G}$ to be bisected, the* closure $\operatorname{clos}_{\mathcal{G}}^{p,q}(\mathcal{M})$ *of $\mathcal{M}$ is computed as follows.*

$\widetilde{\mathcal{M}} := \mathcal{M}$
repeat
  for all $K \in \widetilde{\mathcal{M}}$ ***do***
    $\widetilde{\mathcal{M}} := \widetilde{\mathcal{M}} \cup \big\{K' \in \mathcal{G}^{p,q}(K) \mid \ell(K') < \ell(K)\big\}$
  end for
until $\widetilde{\mathcal{M}}$ *stops growing*
return $\operatorname{clos}_{\mathcal{G}}^{p,q}(\mathcal{M}) = \widetilde{\mathcal{M}}$

Algorithm 2 (Refinement) *Given a mesh $\mathcal{G}$ and a set of marked elements $\mathcal{M} \subseteq \mathcal{G}$ to be bisected,* $\operatorname{ref}^{p,q}(\mathcal{G},\mathcal{M})$ *is defined by*

$$\operatorname{ref}^{p,q}(\mathcal{G},\mathcal{M}) := \operatorname{bisect}(\mathcal{G}, \operatorname{clos}_{\mathcal{G}}^{p,q}(\mathcal{M})).$$

Definition 7 (admissible bisections) *Given a mesh $\mathcal{G}$ and an element $K \in \mathcal{G}$, the bisection of K is called* admissible *if all $K' \in \mathcal{G}^{p,q}(K)$ satisfy $\ell(K') \ge \ell(K)$.*

In the case of several elements $\mathcal{M} = \{K_1,\dots,K_J\} \subseteq \mathcal{G}$, the bisection $\operatorname{bisect}(\mathcal{G},\mathcal{M})$ *is admissible if there is an order $(\sigma(1),\dots,\sigma(J))$ (this is, if there is a permutation σ of $\{1,\dots,J\}$) such that*

$$\operatorname{bisect}(\mathcal{G},\mathcal{M}) = \operatorname{bisect}(\dots \operatorname{bisect}(\mathcal{G},K_{\sigma(1)})\dots,K_{\sigma(J)})$$

is a concatenation of admissible bisections.

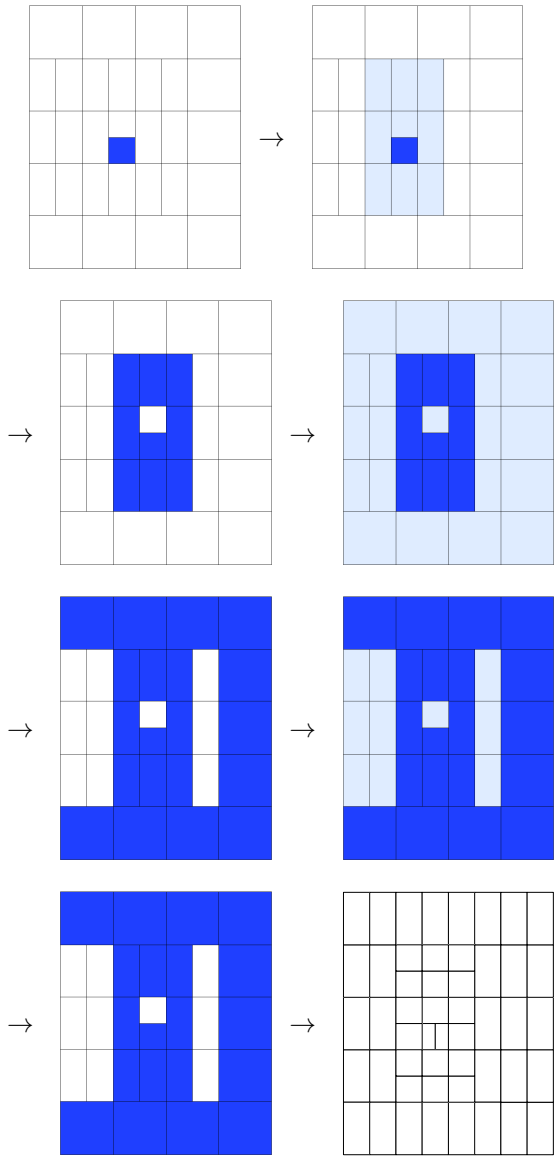


Figure 2: Algorithm 1 recursively marks coarser elements in the patch of the initially marked K. In this case, the computation of $\mathrm{clos}_{\mathcal{G}}^{p,q}(\{K\})$ involves three iterations of the algorithm.

Definition 8 (Admissible mesh) *A refinement $\mathcal{G}$ of $\mathcal{G}_0$ is* admissible *if there is a sequence of meshes $\mathcal{G}_1, \dots, \mathcal{G}_J = \mathcal{G}$ and markings $\mathcal{M}_j \subseteq \mathcal{G}_j$ for $j = 0, \dots, J-1$, such that $\mathcal{G}_{j+1} = \mathrm{bisect}(\mathcal{G}_j, \mathcal{M}_j)$ is an admissible bisection for all $j = 0, \dots, J-1$. The set of all admissible meshes, which is the initial mesh and its admissible refinements, is denoted by $\mathbb{A}^{p,q}$.*

Proposition 1 *Any admissible mesh $\mathcal{G}$ and any set of marked elements $\mathcal{M} \subseteq \mathcal{G}$ satisfy that $\mathrm{ref}^{p,q}(\mathcal{G}, \mathcal{M}) \in \mathbb{A}^{p,q}$.*

Altogether, $\mathbb{A}^{p,q}$ is the set of meshes that are generated by Algorithm 2.

3 Main results

This section discusses the coarsest common refinement of two meshes $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{A}^{p,q}$, called *overlay* and denoted by $\mathcal{G}_1 \otimes \mathcal{G}_2$. Subsequently, we give results on analysis-suitability, nestedness, and complexity.

Definition 9 (Overlay) *We define the operator $\mathrm{Min}_{\subseteq}$ which yields all minimal elements of a set that is partially ordered by "$\subseteq$",*

$$\mathrm{Min}_{\subseteq}(\mathcal{M}) := \big\{ K \in \mathcal{M} \mid \forall K' \in \mathcal{M} : K' \subseteq K \Rightarrow K' = K \big\}.$$

The overlay *of $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{A}^{p,q}$ is defined by*

$$\mathcal{G}_1 \otimes \mathcal{G}_2 := \mathrm{Min}_{\subseteq}\big(\mathcal{G}_1 \cup \mathcal{G}_2 \big).$$

Proposition 2 *For any admissible meshes $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{A}^{p,q}$, the overlay $\mathcal{G}_1 \otimes \mathcal{G}_2$ is also admissible.*

Lemma 3 *For all $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{A}^{p,q}$ holds*

$$\#\left(\mathcal{G}_1 \otimes \mathcal{G}_2\right) + \#\mathcal{G}_0 \leq \#\mathcal{G}_1 + \#\mathcal{G}_2 \ .$$

Theorem 4 *All admissible meshes (in the sense of Definition 8) are analysis-suitable.*

Corollary 5 *All admissible meshes provide T-spline blending functions that are non-negative, linearly independent, and form a partition of unity [6, 7]. Moreover, on each element $K \in \mathcal{G} \in \mathbb{A}^{p,q}$, there are not more than $2(p+1)(q+1)$ T-spline basis functions that have support on K [7, Proposition 7.6].*

This means that on each element, each T-Spline function communicates only with a finite number of other T-spline functions, independent of the total number of functions. This is an important requirement for sparsity of the linear system to be solved in Finite Element Analysis, in the sense that every row and every column of a corresponding stiffness or mass matrix is a sparse vector.

Theorem 6 *For any two meshes $\mathcal{G}_1, \mathcal{G}_2 \in \mathbb{A}^{p,q}$ that are nested in the sense $\mathcal{G}_1 \preceq \mathcal{G}_2$, the corresponding T-spline spaces are also nested.*

Theorem 7 *Any sequence of admissible meshes $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_J$ with $\mathcal{G}_j = \mathrm{ref}^{p,q}(\mathcal{G}_{j-1}, \mathcal{M}_{j-1})$ and $\mathcal{M}_{j-1} \subseteq \mathcal{G}_{j-1}$ for $j \in \{1, \ldots, J\}$ satisfies*

$$|\mathcal{G}_J \setminus \mathcal{G}_0| \;\leq\; C_{p,q} \sum_{j=0}^{J-1} |\mathcal{M}_j|,$$

with $C_{p,q}$ depending only on p and q.

Theorem 7 shows that, with regard to possible mesh gradings, the refinement algorithm is as flexible as successive bisection without the closure step. However, this result is non-trivial. Given a mesh $\mathcal{G} \in \mathbb{A}^{p,q}$ and an element $K \in \mathcal{G}$ to be bisected, there is no uniform bound on the number of generated elements $\#(\mathrm{ref}^{p,q}(\mathcal{G}, \{K\}) \setminus \mathcal{G})$.

4 Conclusion

We presented an adaptive refinement algorithm for a subclass of analysis-suitable T-meshes that produces nested T-spline spaces, and we proved theoretical properties that are crucial for the analysis of adaptive schemes driven by a posteriori error estimators. As an example, compare the assumptions (2.9) and (2.10) in [2] to Theorem 7 and Lemma 3, respectively. The presented refinement algorithm can be extended to the three-dimensional case, which is our current work. The factor $C_{p,q}$ from the complexity estimate is affine in each of the parameters p, q and increases exponentially with growing dimension. We aim to apply the proposed algorithm to proof the rate-optimality of an adaptive algorithm for the numerical solution of second-order linear elliptic problems using T-splines as ansatz functions. Similar results have been proven for simple FE discretizations of the Poisson model problem in 2007 by Stevenson [15], in 2008 by Cascon, Kreuzer, Nochetto and Siebert [3], and recently for a wide range of discretizations and model problems by Carstensen, Feischl, Page and Praetorius [2].

References

[1] A. Buffa, D. Cho, and G. Sangalli. Linear independence of the T-spline blending functions associated with some particular T-meshes. *Comput. Methods Appl. Mech. Engrg.*, 199(2324):1437 – 1445, 2010.

[2] C. Carstensen, M. Feischl, M. Page, and D. Praetorius. Axioms of adaptivity. *Comput. Math. Appl.*, 67(6):1195–1253, 2014.

[3] J. Cascon, C. Kreuzer, R. Nochetto, and K. Siebert. Quasi-Optimal Convergence Rate for an Adaptive Finite Element Method. *SIAM J. Numer. Anal.*, 46(5):2524–2550, 2008.

[4] J. Cottrell, T. Hughes, and Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*, pages i–xvi. John Wiley & Sons, Ltd, 2009.

[5] L. B. da Veiga, A. Buffa, D. Cho, and G. Sangalli. Analysis-Suitable T-splines are Dual-Compatible. *Comput. Methods Appl. Mech. Engrg.*, 249-252:42–51, 2012. Higher Order Finite Element and Isogeometric Methods.

[6] L. B. da Veiga, A. Buffa, G. Sangalli, and R. Vàzquez. Analysis-suitable T-splines of arbitrary degree: definition, linear independence and approximation properties. *Math. Models Methods Appl. Sci.*, 23(11):1979–2003, 10 2013.

[7] L. B. da Veiga, A. Buffa, G. Sangalli, and R. Vàzquez. Mathematical analysis of variational isogeometric methods. *Acta Numerica*, 23:157–287, 5 2014.

[8] T. Hughes, J. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engrg.*, 194(3941):4135 – 4195, 2005.

[9] X. Li and M. A. Scott. Analysis-suitable t-splines: Characterization, refineability, and approximation. *Math. Models Methods Appl. Sci.*, 24(06):1141–1164, 2014.

[10] X. Li, J. Zheng, T. Sederberg, T. Hughes, and M. Scott. On Linear Independence of T-spline Blending Functions. *Comput. Aided Geom. Des.*, 29(1):63–76, Jan. 2012.

[11] P. Morgenstern and D. Peterseim. Analysis-suitable adaptive T-mesh refinement with linear complexity. *ArXiv e-prints*, July 2014. Submitted for publication.

[12] M. Scott, X. Li, T. Sederberg, and T. Hughes. Local refinement of analysis-suitable t-splines. *Comput. Methods Appl. Mech. Engrg.*, 213216:206–222, 2012.

[13] T. Sederberg, D. Cardon, G. Finnigan, N. North, J. Zheng, and T. Lyche. T-spline Simplification and Local Refinement. *ACM Trans. Graph.*, 23(3):276–283, Aug. 2004.

[14] T. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-Splines and T-NURCCs. *ACM Trans. Graph.*, 22(3):477–484, July 2003.

[15] R. Stevenson. Optimality of a standard adaptive finite element method. *Found. Comput. Math.*, 7(2):245–269, 2007.

The Slope Number of Segment Intersection Graphs

Udo Hoffmann *

Abstract

We prove that it is NP-hard to determine the minimal number of slopes that is required to draw a segment representation of a segment intersection graph. As a side product we obtain new proofs for the NP-hardness of the recognition of grid, segment and pseudosegment graphs. We also show, that the minimum number of slopes of a segment graph can drop arbitrarily upon the removal of a single vertex.

1 Introduction

Intersection graphs of segments are known to have complex geometric and combinatorial properties. The recognition is NP-hard [1] and even complete for the existential theory of the reals [2]. Kratochvíl and Matoušek [8] show that the complexity of this problem decreases if the number of slopes allowed for a segment representation is bounded by a constant. Then, for every fixed $k \geq 2$ the problem of deciding whether a graph is the intersection graph of segments using k different slopes is NP-complete [3].

The slope number of segment graphs is also of interest for planar graphs. Planar bipartite graphs are known to be contact graphs of segments with two slopes [4], while de Castro *et al.* [5] show that each K_3-free planar graph can be represented as a segment contact graph of segments using three slopes. This result is supported by a result of Grötzsch , who showed that a K_3-free planar graph is 3-colorable. Chalopin and Gonçalves [6] show that each planar graph can be represented as segment intersection graph. Assuming a representation of a segment intersection graph without two intersecting segments of the same slope, the only lower bound on the slope number of planar graphs is four by the four color theorem.

Our main result is that the minimization of the slope number is NP-hard. As side product we obtain new proofs which show that the recognition of segment, pseudosegment, and grid intersection graphs is NP-hard. The new reduction complements previous ones, since it distinguishes between (pseudo)segment and grid intersection graphs, while the proof of Kratochvíl [3] gives graphs which are a grid intersection graph iff they are a (pseudo)segment intersection graph.

We also show that the number of slopes does not behave well under deleting or adding vertices. There are segment graphs using a linear number of slopes in terms of their vertices, but the removal of one vertex leads to a segment representation using only two slopes.

In the rest of this section we introduce the notations. In Section 2 we present our main observation: A connection between Hamiltonian paths in planar graphs and segment representations of an associated graph. In Section 3 we show that there are bipartite graphs that require a high number of slopes, and that removing one segment leaves a grid intersection graph. The hardness results for minimizing the number of slopes, which also follow directly from Section 2, are shown in Section 4.

1.1 Notation

An *intersection representation* of a graph is an assignment of a set to each vertex, such that two vertices are adjacent if and only if the sets intersect. For simplicity we do not distinguish between vertices and sets, e.g. we say the vertex v intersects the vertex w when we talk about intersection graphs. We study the following classes of intersection graphs.

The class of *segment graphs*, i.e., the intersection graphs of segments in the plane, is denoted by *SEG*. We consider *pure* representations of segment intersection graphs, i.e., we forbid intersections of parallel segments. Segment intersection graphs using only two slopes (without loss of generality vertical and horizontal) are also known as *grid intersection graphs* (GIG). The intersection graph of *pseudosegments*, i.e., of curves, such that each pair of curves intersects at most once, is denoted by *pSEG*.

We will assume that all segments have positive slopes. This can be achieved by a suitable linear transformation of the plane.

A *k-page book embedding*, is a linear order L of the vertices and a partition of the edges into k sets, such that each set can be drawn without crossings in a half plane, where the vertices are drawn on the boundary in the order given by L. A 1-page book embedding exists iff the graph is outerplanar. Here the linear order L corresponds to a possible order of the vertices in the outer face in an outerplanar drawing.

*uhoffman@math.tu-berlin.de, Technische Universität. Supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

Let $G = (V, E)$ be a graph. Denote its full subdivision by P_G, i.e., the graph $P_G := (V \cup E, \{ve | v \in e \in E\})$. Let $P_G^* := (V \cup E \cup \{e^*\}, E(P_G) \cup \{ve^* | v \in V\})$ be the full subdivision of P_G where one vertex that is incident to every original vertex is added.

We are interested in segment and GIG representations of P_G^*. It is well known that P_G admits a GIG representation if and only if G is planar, for example as bar visibility representations in [11]. A *semibar visibility* representation of a graph is a representation of vertices as vertical segments with the same y-coordinate of the lower endpoint. The edges are represented as horizontal sightlines.

Lemma 1 ([12, 7]) *A graph G has a semibar visibility representation iff it has a 1-page book embedding. This is the case iff G is outerplanar. All possible orders of the vertices in a representation are given by the order of the vertices of an outer face cycle of G.*

1.2 Results

The main tools used in the paper are the following two theorems which are proven in Section 2.

Theorem 2 *A graph G has a 2-page book embedding iff P_G^* is a GIG.*

It is known that a graph has a 2-page book embedding iff it is a subgraph of a planar graph with Hamiltonian cycle [7]. We give a similar condition for (pseudo)segment representations of P_G^*.

Theorem 3 *For a graph G, the graph P_G^* is a (pseudo)segment intersection graph iff G is a subgraph of a planar graph with Hamiltonian path.*

The graph class described by the theorem above are known as *deque graphs* [9].

The characterizations in Theorem 2 and Theorem 3 allow the translation of the following lemma to Theorem 5.

Lemma 4 *The Hamiltonian cycle problem in maximal planar graphs with given Hamiltonian path is NP-hard.*

Theorem 5 *The recognition of grid intersection graphs is NP-complete, even if a segment representation (using four slopes) is given.*

In Section 3, we show that the number of slopes can decrease drastically by removing only one vertex.

Theorem 6 *There is a family of graphs G_n, $n \in \mathbb{N}$, with $|V(G_n)| = n$, such that each segment representation of G_n requires $\Theta(n)$ slopes, but G_n has a vertex v, such that $G_n - v$ has a GIG representation.*

2 Segment intersection graphs and Hamiltonian paths

In this section we give a proof for Theorem 2 and Theorem 3.

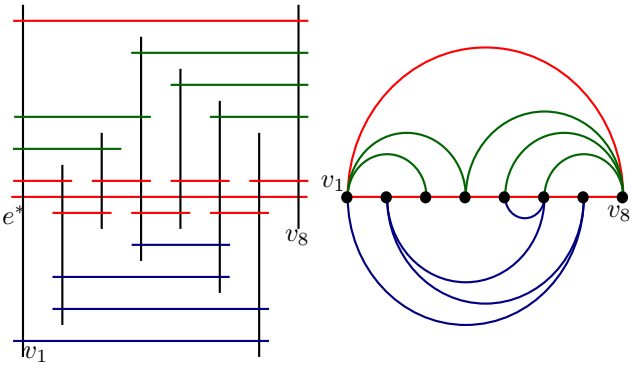


Figure 1: A GIG rep. of P_G^* and a 2-page book embedding of G.

Proof of Theorem 2. Consider a GIG representation of P_G^* as in Figure 1. The representation above e^* can be modified to a semibar visibility representation by shortening the edges to contacts. This drawing can be replaced by a 1-page book embedding with the same order of the vertices. The same modification for the edges below e^* leads to a 2-page book embedding.

On the other hand, the two 1-page book embeddings can be replaced by equivalent semibar visibility representations, such that a 2-page book embedding leads to a GIG representation of P_G^*. □

We give an idea of the proof of Theorem 3. We first argue that a pseudosegment representation of P_G^* gives a Hamiltonian path in a planar graph G' which has G as subgraph. Consider a (pseudo)segment representation of P_G^*. The order of intersections on e^* gives the order of the vertices on a Hamiltonian path. Let $v_1, \ldots, v_n$ be the order of the vertices according to their intersection with e^*. If $v_i v_{i+1} \in E(G)$ there is nothing to show. Otherwise we draw a pseudosegment 'parallel' to e^* that intersects only v_i and v_{i+1}. This way we obtain a representation of $P_{G'}^*$ where G' is a planar graph with Hamiltonian path $v_1, \ldots, v_n$ and G as subgraph.

Now we show that a planar graph G' with Hamiltonian path $v_1, \ldots, v_n$ admits a pseudosegment representation of $P_{G'}^*$.

Let G' be a planar graph with Hamiltonian path $v_1, \ldots, v_n$. We construct a pseudosegment representation of $P_{G'}^*$ from a planar drawing of G'. Consider a planar drawing of G' with small circles as vertices and pseudosegments as edges, that have contacts on its incident vertices.

We draw e^* as curve that intersects only the vertices in the order of the Hamiltonian path. The vertices

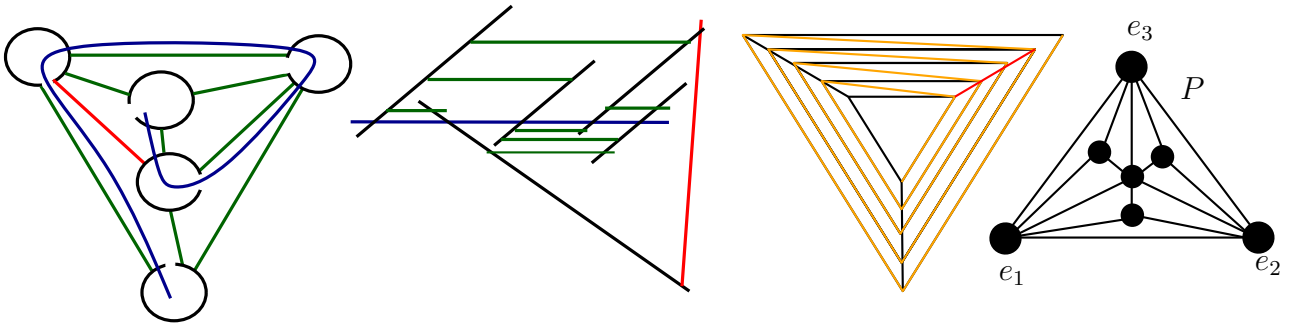


Figure 2: A pseudosegment representation with blue Hamiltonian path and red bad edge and its segment representation. A graph with orange Hamiltonian path with a long chain of bad edges (red). A gadget to fix a certain Hamiltonian path.

are still drawn as cycles and the Hamiltonian path intersects twice with each vertex v_i, $2 \leq i < n$. So opening the small circles removing a segment of v_i, that contains only one intersection with e^*, leads to a pseudosegment representation of $P^*_{G'}$, see Figure 2. □

We omit a full proof for the fact that P^*_G is also a segment intersection graph.

3 Number of slopes

We continue to examine the structure of the bad edges to give a lower bound on the number of slopes used for a segment intersection representation of P^*_G. In the following we assume that e^* is represented by a horizontal segment.

In a segment representations of P^*_G, edges between the upper part (above e^*) of one vertex and the lower part (below e^*) of another vertex are possible as shown Figure 2. We call such an edge a *bad edge*. We call a bad edge v_iv_j, $i < j$, an *upwards* edge if it connects the lower part of v_i and the upper part of v_j and a *downwards* edge otherwise. Denote the closed curve on the pseudosegments v_i, v_j, v_iv_j and e^* by C_{ij}. If v_iv_j is a bad edge, then the bounded region of C_{ij} contains exactly one of the ends of e^*. If it contains the end close to v_1 we call v_iv_j a *front* edge, otherwise a *back* edge. The choice of the outer face of G determines (for a fixed Hamiltonian path) which bad edge is a front resp. a back edge.

We fix a segment representation of P^*_G. Let $w_iw_{i+1}, w_{i+1}w_{i+2}, \ldots, w_{i+k-1}w_{i+k}$ be bad edges of the same upwards/downwards-type. We call the vertices a *chain of length* $k+1$. If all the edges of the chain are front or all back edges it is called a *monotone* chain. The length of the longest chain gives a lower bound on the number of slopes needed in a segment representation with this fixed Hamiltonian path.

Lemma 7 *Let p be a Hamiltonian path in G and C be the longest chain. Any representation of P^*_G whose vertex order is given by p requires $2\left\lceil \frac{|C|}{2} \right\rceil$ slopes.*

Proof. Consider a segment representation of P^*_G. For a bad front edge we know that the segment with the upper part has a larger slope then the segment with the lower part (otherwise they can only be connected by a back edge), and the opposite holds for back edges. The bad edges connecting vertices in the chain require slopes that lie between the slopes of their incident vertices. On the other hand, one chain can be divided into two monotone chains of front and back edges by the choice of the outer face of G. At least one of the monotone chains of front or back edges has length $l := \left\lceil \frac{|C|}{2} \right\rceil$. P^*_G requires l slopes for the vertices of the monotone chain, $l-1$ for the edges between, and one for e^*. □

We show that there is a triangulation that has only Hamiltonian paths including long chains. This way, we prove that there are graphs G such that P^*_G requires a large number of slopes in any segment representation.

Lemma 8 *Let G be a planar graph with Hamiltonian path $p = v_1, \ldots, v_n$. There exists a planar graph G' with $\Theta(|V(G)|)$ vertices, such that G is an induced subgraph of G' and every Hamiltonian path in every triangulation of G' visits at least half the vertices of G in the same order (up to the reverse) as p.*

Proof. Consider the graph P in Figure 2. This graph has the property that every path from e_1 to e_3 using all inner vertices also uses e_2. Hence for two copies P_1, P_2 of P, where two outer vertices are identified every Hamiltonian path visits P_1, P_2 consecutively, or has one end in each. Now, glueing a copy of P on every edge of p, i.e., we identify the edge v_iv_{i+1} of p with e_1e_3 of a copy of P, leads to a graph G' with the desired property. □

Proof of Theorem 6. Consider the graph G_n with $3n$ vertices in Figure 2 with the drawn Hamiltonian

path p (orange), i.e., nested triangles with spiraling Hamiltonian path. G_n has a chain of length n (red). Constructing G'_n from Lemma 8 gives a graph, such that $P^*_{G'_n}$ requires $\lceil \frac{n}{4} \rceil$ different slopes in any segment representation. G'_n and hence $P^*_{G'_n}$ has $\Theta(n)$ vertices. Removing e^* leads to a grid intersection graph. □

4 Computational complexity

The complexity of the recognition of (pseudo)segment and grid intersection graphs is NP-hard by the facts that the Hamiltonian cycle and Hamiltonian path problem in maximal planar graphs are NP-hard [10].

That the minimization of the required number of slopes of a segment intersection graphs is NP-hard follows from the Theorem 5.

The idea of the proof Lemma 4 is the following. A gadget of from [10] transforms a planar cubic bipartite graph G into a maximal planar graph T which has a Hamiltonian cycle iff G has.

Now we are interested in 3SAT instances that lead to a maximal planar graph with Hamiltonian path in the reduction. A class of 3SAT instances satisfying this property is the following. Consider a 3SAT instance I with a truth assignment A, such that A satisfies all but one clause of I. It turns out that a Hamiltonian path in the triangulation can be constructed using A. We call those instances *almost satisfiable*.

Lemma 9 *Deciding if an almost satisfiable 3SAT instance has a satisfying truth assignment is NP-complete.*

Proof. We give a reduction from the 3SAT problem. Given a 3SAT instance, we take all clauses and modify them in the following way. We replace each literal in each clause with a new variable, so each variable appears only once. In addition we introduce a variable x and the clause $\overline{x}$. For each pair of variables y_1, y_2, where both corresponded to a literal of the same variable,we add the clauses $x \vee y_1 \vee \overline{y_2}$ and $x \vee \overline{y_1} \vee y_2$ to make them equivalent if they were both non-negated or both negated, or $x \vee y_1 \vee y_2$ and $x \vee \overline{y_1} \vee \overline{y_2}$ if one literal was negated. The constructed 3SAT is satisfiable iff the original is: In a satisfying truth assignment x is false, hence the introduced clauses give the equivalence between y_1 and y_2, or their negation respectively. This results in a satisfying truth assignment for the original 3SAT. On the other hand, a truth assignment satisfying all but one clause is given by setting x true and all other variables true. The size of the new 3SAT is polynomial in the size of the original one. □

The Hamiltonian path in a resulting triangulation of the reduction can be constructed such that both ends of the path lie in the gadget corresponding to the variable x, such that x is in 'superposition' and the path treats x as true and false simultaneously. We omit the details here due to space limitations.

Acknowledgments Many thanks to Stefan Felsner, Steve Chaplick and Linda Kleist for helpful discussions.

References

[1] J. Kratochvíl and J. Matoušek NP-hardness results for intersection graphs. *Comm. Math. Univ. Caro.*, 30(4):761–773, 1989.

[2] P.W. Shor Stretchability of pseudolines is NP-hard *Appl. Geom. a. Discr. Math.*, 4:373–382, 1991.

[3] J. Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness *Discrete and Applied Mathematics*, 52(3):233-252, 1994.

[4] H. de Fraysseix and P.O. de Mendez and J. Pach. Representation of planar graphs by segments. *Intuitive Geometry*, 63:109–117, 1991.

[5] F.J. de Castro and N. Cobos and J.C. Dana and A. Márquez and M. Noy. Triangle-Free Planar Graphs as Segment Intersection Graphs. *Graph Algorithms and Applications*, 6(1):7–26, 2004.

[6] J. Chalopin and D. Gonçalves. Every planar graph is the intersection graph of segments in the plane. *Proc. annual ACM Symposium on Theory of Computing*, 50(2):631–638, 2009.

[7] F.R. Bernhart and P.C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B.* 27(3):320–331, 1979

[8] J. Kratochvíl and J. Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B.* 62(2):289–315, 1994.

[9] C. Auer and A. Gleißner. Characterizations of deque and queue graphs. *Graph-Theoretic Concepts in Computer Science.* 35–46:289–315, 2011.

[10] A. Wigderson. The complexity of the Hamiltonian circuit problem for maximal planar graphs. Technical Report, Princeton University, 1982.

[11] S.K. Wismath. Characterizing bar line-of-sight graphs. *Proc. SoCG*, 147–152, 1985.

[12] F.J. Cobos and J.C. Dana and F. Hurtado and A. Márquez and F. Mateos. On a visibility representation of graphs. *Proc. Graph Drawing*, 1027:152–161, 1996.

Recognizing Weighted Disk Contact Graphs

Boris Klemz* Martin Nöllenburg* Roman Prutkin*

Abstract

Disk contact representations realize graphs by mapping vertices to interior-disjoint disks in the plane such that disks touch each other if and only if the corresponding vertices are adjacent. Deciding whether a vertex-weighted graph can be realized so that the disks' radii coincide with the vertex weights has been proven NP-hard. In this work, we analyze the problem for special graph classes and show that it remains hard even for very basic ones, thereby strengthening previous NP-hardness results. On the positive side, we present linear-time algorithms for two restricted versions of the problem.

1 Introduction

A disk intersection representation is a set of disks in the plane that can be interpreted as a graph containing a vertex for each of its disks and an edge for each pair of intersecting disks. *Disk intersection graphs* are graphs that have a disk intersection representation and generalize *disk contact graphs*, that is, graphs that have a disk intersection (or contact) representation with interior-disjoint disks. Koebe's Theorem [9] is a classic result in graph theory that states that any planar graph is a disk contact graph, and for any disk contact representation it is easy to obtain a planar drawing of the realized graph.

Application areas for disk intersection/contact graphs include modeling physical problems like wireless communication networks [6], covering problems like geometric facility location [10, 11], visual representation problems like the generation of area cartograms [4] and many more (various examples are given by Clark et al. [3]). Often, one is interested in recognizing disk graphs or generating representations that do not only realize the input graph, but also satisfy additional requirements. For example, Alam et al. [1] recently studied graphs having disk contact representations, in which the ratio of the largest disk radius to the smallest is polynomial in the number of disks. Furthermore, it might be desirable to generate a disk representation that realizes a vertex-weighted graph such that the disk radii or areas reflect the corresponding vertex weights, for example, for value-by-area circle cartograms [7]. Clearly, there exist vertex-weighted planar graphs that can not be realized as disk contact graphs, and the corresponding recognition problem is NP-hard even if all vertices are weighted uniformly [2].

We examine the aforementioned scenario more closely and explore disk contact representations for special graph classes. We show that it is NP-hard to decide whether a realization with uniform radii exists even if the input graph is outerplanar and even if a combinatorial embedding is provided. On the other hand, we can decide in linear time whether a caterpillar is a disk contact graph with uniform disk radii. If the vertex weights are not necessarily uniform, the recognition problem becomes NP-hard even for stars, but it can be solved in linear time for a given combinatorial embedding.

2 Unit disk contact graphs

In this section we are concerned with the problem of deciding whether a given graph is a unit disk contact (UDC) graph, that is, whether it can be realized as a unit disk contact representation. It is known that this is generally NP-hard for planar graphs [2], but it remained open for which subclasses of planar graphs the realizability problem can be efficiently decided and for which subclasses NP-hardness still holds.

We show that for caterpillars we can decide the realizability problem (and construct a representation if it exists) in linear time, whereas the problem remains NP-hard for outerplanar graphs.

Recognizing realizable caterpillars. Let $G = (V, E)$ be a caterpillar graph, that is, a tree for which a path remains after removing all leaves. Let $P = (v_1, \dots, v_k)$ be this so-called *inner path* of G. It is well known that six unit disks can be tightly packed around one central unit disk, but then any two consecutive outer disks necessarily touch and form a triangle with the central disk. This is not permitted in a caterpillar and we obtain that in any realizable caterpillar the maximum degree $\Delta \leq 5$. For $\Delta \leq 4$ it is easy to see that G can always be realized as shown in Fig. 1.

However, not all caterpillars with $\Delta = 5$ can be realized. For example, two degree-5 vertices on P separated by zero or more degree-4 vertices cannot be realized, as they would again require tightly packed disks inducing cycles in the contact graph.

*Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany. boris.klemz@student.kit.edu, noellenburg@kit.edu, roman.prutkin@kit.edu

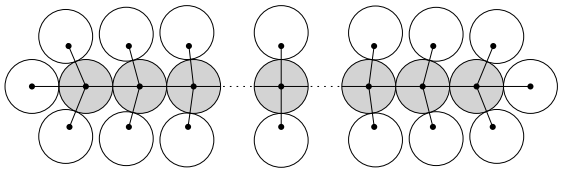


Fig. 1: For $\Delta \leq 4$ any caterpillar can be realized.

It turns out that a simple iterative pass along P suffices to decide the realizability of G and find a realization if possible. We place a disk D_1 for v_1 at the origin and attach its leaf disks *leftmost*, that is, symmetrically pushed to the left with a sufficiently small distance between them. In each subsequent step, we place the next disk D_i for v_i on the bisector of the *free space*, i.e. the maximum cone with origin in D_{i-1}'s center containing no previously inserted neighbors of D_{i-1} or D_{i-2} and attach the leaves of D_i in a leftmost and balanced way, see Fig. 2. For odd numbers of leaves this leads to a change in direction of P, but by alternating upward and downward bends for subsequent odd-degree vertices we can maintain a horizontal monotonicity, which ensures that leaves of D_i can only collide with leaves of D_{i-1} and D_{i-2}. If we fail to place the disks correctly, we claim that no UDC representation of G exists; otherwise we return the constructed realization.

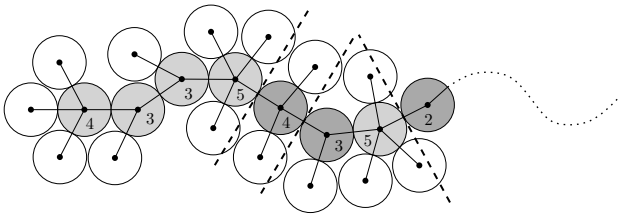


Fig. 2: Incremental construction of a realization. Narrow disks are dark gray, wide disks are light gray.

For a sketch of correctness, consider the tangent line ℓ_i between two adjacent disks D_{i-1} and D_i on the inner path. We say that P is *narrow* at v_i if some leaf disk attached to D_{i-1} intersects ℓ_i; otherwise P is *wide* at v_i. We observe that in our construction P gets narrow precisely when a degree-5 vertex of P is encountered. But it is generally true in any representation that P gets narrow after a degree-5 vertex. If P is narrow at v_i this means that at most three disjoint disks touching D_i can still be placed and thus it must be $\deg(v_i) \leq 4$. Each vertex of degree 4 inherits the narrow/wide status of its predecessor. Vertices with degree 3 or less make P wide again.

This idea leads to a combinatorial characterization (and decision algorithm) of caterpillars with a UDC representation, based on the property that between any two degree-5 vertices on P there must be at least one vertex of degree at most 3.

Theorem 1 *For a caterpillar G it can be decided in linear time whether G is a UDC graph. A realization (if one exists) can be constructed in linear time on a Real RAM.*

Hardness for outerplanar graphs. We perform a polynomial reduction from the classic NP-complete 3SAT problem to show NP-hardness of the UDC-realizability problem for outerplanar graphs. Here, we sketch only the main ideas of the reduction and refer to Klemz [8, Chapter 2] for more details.

Let φ be a Boolean 3SAT formula with a set $\mathcal{U} = \{x_1, \ldots, x_n\}$ of n variables and a set $\mathcal{C} = \{c_1, \ldots, c_m\}$ of m clauses, where each c_i contains three literals over $\mathcal{U}$. We create the *literal-clause-graph* $G_\varphi = (\mathcal{U} \cup \bar{\mathcal{U}} \cup \mathcal{C}, E)$, where $\bar{\mathcal{U}}$ is the set of negative literals over $\mathcal{U}$. The set E contains for each clause $c \in \mathcal{C}$ the edge (c, x) if literal x occurs in c and the edge $(c, \bar{x})$ if literal $\bar{x}$ occurs in c. Based on G_φ we create an outerplanar graph G'_φ that has a UDC representation if and only if the formula φ is satisfiable.

Arguing about UDC realizations of certain subgraphs becomes a lot easier, if there is only a single unique geometric representation (up to rotation, translation and mirroring). We call such a representation *rigid*. Using an inductive argument, we can show the following lemma about rigid UDC structures.

Lemma 2 *A unit disk contact representation whose UDC graph is biconnected, internally triangulated and outerplanar is rigid.*

The main building block of the reduction is a *wire gadget* in G'_φ that comes in different variations but always consists of a rigid tunnel structure containing a rigid bar that can be flipped into different tunnels around its centrally located articulation vertex. Each wire gadget occupies a square tile of fixed dimensions so that different tiles can be flexibly put together in a grid-like fashion. The bars stick out of the tiles in order to transfer information to the neighboring tiles. Some special tiles of the variable gadgets consist of tunnels without bars or with very long bars. Finally, we construct *crossing gadgets* that correctly transmit information along both axes of a tunnel crossing. Figure 3 shows a schematic view of how the gadget tiles are arranged to form a layout of G_φ.

The main idea behind the reduction is as follows. Each variable gadget contains one long horizontal bar that is either flipped to the left (*false*) or to the right (*true*), see Fig. 4(b). Consequently, each wire gadget of a literal edge connecting a variable gadget to a clause gadget must flip its chain of bars towards the clause if the literal is false. Finally, each clause gadget has one central T-shaped wire gadget, whose bar needs to be placed inside one of the three incoming

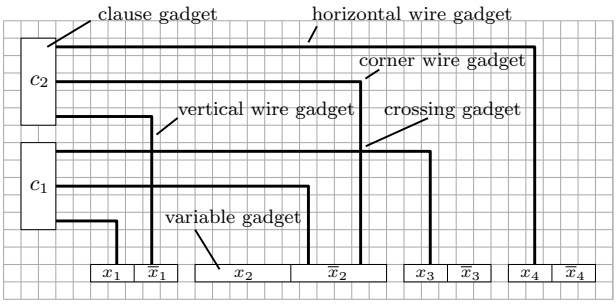


Fig. 3: High-level structure of the construction for the 3SAT formula $\varphi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$.

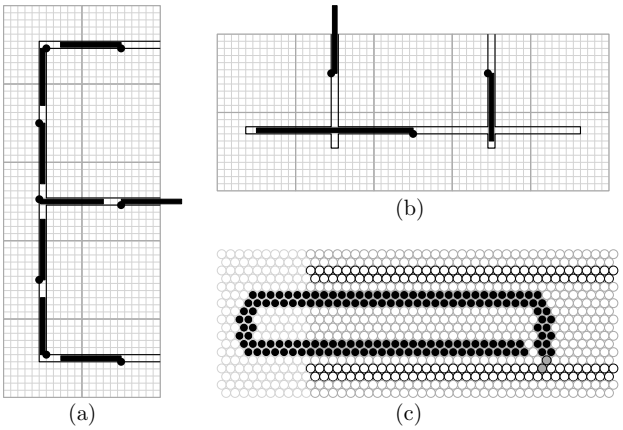


Fig. 4: (a) Clause gadget with two false inputs (top and bottom) and one true input, (b) variable gadget in the state *false* with one positive (left) and one negative literal (right), (c) detailed view of a horizontal wire gadget with a rigid bar (black disks) inside a horizontal tunnel (white disks).

tunnels. This is possible if and only if at least one of the literals evaluates to *true*, see Fig. 4(a).

Clearly, all gadgets need to be realized by rigid unit disk contacts. Figure 4(c) shows a close-up of a horizontal wire gadget. The position of the bars inside the tunnels admits some slack, but it does not affect the combinatorial properties.

Finally, one needs to take care that the constructed graph is actually outerplanar and connected. This is not obvious, but can be done by introducing small gaps and a modification in the attachment of the bar in some of the horizontal wire gadgets. Moreover, the reduction can be further modified so that it remains valid for outerplanar graphs with a fixed embedding; details can be found in [8].

Theorem 3 *The UDC graph recognition problem is* NP*-hard, even for outerplanar graphs and even if a combinatorial embedding is given.*

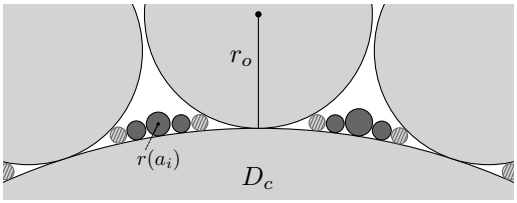


Fig. 5: Reducing from 3-Partition to prove Theorem 4. Input disks (dark) have to be distributed between gaps. Striped disks are separators.

3 Weighted disk contact graphs

In this section, we assume that each graph vertex has a positive weight, which corresponds to the disk radius of the representing disk. Deciding whether a weighted disk contact (WDC) representation respecting this radius assignment exists is obviously at least as hard as the UDC problem from Section 2.

Hardness for stars. Compared to Section 2, for an arbitrary radius assignment the corresponding recognition problem is hard for even simpler graph classes.

We perform a polynomial reduction from the well-known 3-Partition problem. Given a bound $B \in \mathbb{N}$ and a set of positive integers $\mathcal{A} = \{a_1, \ldots, a_{3n}\}$ such that $\frac{B}{4} < a_i < \frac{B}{2}$ for all $i = 1, \ldots, 3n$, deciding whether $\mathcal{A}$ can be partitioned into n triples of sum B each is known to be strongly NP-complete [5].

Let $(\mathcal{A}, B)$ be a 3-Partition instance. We construct a star S as well as a radius assignment r such that S has a disk contact representation respecting r if and only if $(\mathcal{A}, B)$ is a yes-instance.

We create a central disk D_c of radius r_c corresponding to the central vertex v_c of S as well as a fixed number of outer disks with uniform radius r_o chosen appropriately such that these disks have to be placed close together around D_c without touching, creating funnel-shaped *gaps* of equal size; see Fig. 5. Then, a contact representation exists only if all remaining disks can be distributed among the gaps, and the choice of the gap will induce a partition of the integers $a_i \in \mathcal{A}$. We shall represent each a_i by a single disk called an *input* disk and encode a_i in its radius. Each of the gaps is supposed to be large enough for the input disks that represent a feasible triple to fit inside it, however, the gaps should be too small to contain an infeasible triple's disk representation.

The main challenge is finding a radius assignment satisfying the above property, although numerous additional nontrivial geometric considerations are required to make the construction work. For example, we require that the lower boundary of each gap is sufficiently flat. We achieve this by creating additional *dummy* gaps, which in any realization must be completely filled by special *dummy* disks, such that there

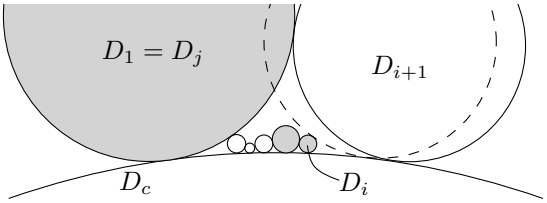


Fig. 6: Deciding existence for Theorem 5. Gray disks are in L before inserting D_{i+1}. After that, the two small gray disks will be removed from L.

are still only n gaps to distribute the input disks. Next, we make sure that additional *separator* disks must be placed in each gap's corners to prevent left and right gap boundaries from interfering with the input disks. Finally, all our constructions are required to tolerate a certain amount of "wiggle room", since, firstly, the outer disks do not touch and, secondly, some radii cannot be computed precisely in polynomial time. Again, we refer to [8, Chapter 3] for a detailed proof.

Theorem 4 *The WDC graph recognition problem is* NP*-hard even for stars.*

Stars with fixed embedding. If the order of the leaves around the central vertex of the star is fixed, the existence of a WDC representation can be decided by tightly placing the outer disks $D_1, \ldots, D_{n-1}$ around the central disk D_c iteratively. By keeping track of possible positions of the next disk, we can achieve $O(n)$ runtime.

Let r_i be the radius of D_i, and assume that D_1 is the largest outer disk. Then, D_2 can be placed next to D_1 clockwise. Suppose we have already added $D_2, \ldots, D_i$. As depicted in Fig. 6, tightly placing D_{i+1} next to D_i might cause D_{i+1} to intersect with a disk inserted earlier, even with D_1. Simply testing for collisions with all previously added disks would yield a total runtime of $O(n^2)$, which we improve to $O(n)$ by keeping a list L of inserted disks which might be relevant for future insertions. Initially, only D_1 is in L. We shall see that L remains sorted by non-increasing radius.

When inserting D_{i+1}, we traverse L backwards and test for collisions with traversed disks, until we find the largest index $j < i$ such that $r_j \in L$ and $r_{i+1} \leq r_j$. Next, we place D_{i+1} tightly next to all inserted disks, avoiding collisions with all traversed disks.

First, note that D_{i+1} cannot intersect disks preceding D_j in L (unless D_{i+1} and D_1 would intersect clockwise, in which case we report non-existence). Next, disks that currently succeed D_j in L will not be able to intersect $D_{i+2}, \ldots, D_{n-1}$ and are therefore removed from L. Finally, we add D_{i+1} to the end of L. Since all but one traversed disks are removed during each insertion, the total runtime is $O(n)$. We report existence if we can insert all disks tightly and there is still space left.

Theorem 5 *On a Real RAM, for a vertex-weighted star S with a given embedding it can be decided in linear time whether S is a WDC graph. A representation respecting the embedding (if one exists) can be constructed in linear time.*

4 Conclusion

We presented hardness results as well as linear-time algorithms for variants of the weighted disk contact graph recognition problem. An interesting open problem is the recognition of trees with a UDC representation. For more results, for example, regarding disk contact representations in which disks have to cover specified points, we refer to Klemz [8].

References

[1] M. Alam, D. Eppstein, M. Goodrich, S. Kobourov, and S. Pupyrev. Balanced circle packings for planar graphs. In *Graph Drawing (GD'14)*, volume 8871 of *LNCS*, pages 125–136. Springer, 2014.

[2] H. Breu and D. G. Kirkpatrick. Unit Disk Graph Recognition is NP-hard. *Computational Geometry*, 9(1-2):3–24, Jan. 1998.

[3] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit Disk Graphs. *Discrete Mathematics*, 86(1–3):165–177, 1990.

[4] D. Dorling. Area Cartograms: Their Use and Creation. In *Concepts and techniques in modern geography*. University of East Anglia: Environmental Publications, 1996.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. W. H. Freeman & Co., 1990.

[6] W. Hale. Frequency Assignment: Theory and Applications. *Proc. IEEE*, 68(12):1497–1514, 1980.

[7] R. Inoue. A New Construction Method for Circle Cartograms. *Cartography and Geographic Information Science*, 38(2):146–152, 2011.

[8] B. Klemz. Weighted Disk Contact Graphs. Diploma thesis, Karlsruhe Institute of Technology, 2014.

[9] P. Koebe. Kontaktprobleme der konformen Abbildung. In *Berichte über die Verhandlungen der Sächsischen Akadmie der Wissenschaften zu Leipzig. Math.-Phas. Klasse*, volume 88, pages 141–164, 1936.

[10] J.-M. Robert and G. Toussaint. Computational Geometry and Facility Location. In *Int. Conf. Operations Research and Management Science*, pages 11–15, 1990.

[11] E. Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *LNCS*, pages 359–370. Springer, 1991.

The Complexity of the Partial Order Dimension Problem – Closing the Gap*

Stefan Felsner[1,a] Irina Mustaţă[1,b] Martin Pergel[2,c]

Abstract. The dimension of a partial order P is the minimum number of linear orders whose intersection is P. There are efficient algorithms to test if a partial order has dimension at most 2. In 1982 Yannakakis [9] showed that for $k \geq 3$ to test if a partial order has dimension $\leq k$ is NP-complete. The height of a partial order P is the maximum size of a chain in P. Yannakakis also showed that for $k \geq 4$ to test if a partial order of height 2 has dimension $\leq k$ is NP-complete. The complexity of deciding whether an order of height 2 has dimension 3 was left open. We show that the problem is NP-complete.

Technically, we show that the decision problem (3DH2) for dimension is equivalent to deciding for the existence of bipartite triangle containment representations (BTCon). This problem allows a reduction from a class of planar satisfiability problems (P-3-CON-3-SAT(4)) which is known to be NP-hard.

1 Introduction

Let $P = (X, \leq_P)$ be a partial order. A linear order $L = (X, \leq_L)$ on X is a *linear extension* of P when $x \leq_P y$ implies $x \leq_L y$. A family $\mathcal{R}$ of linear extensions of P is a *realizer* of P if $P = \bigcap_i L_i$, i.e., $x \leq_P y$ if and only if $x \leq_L y$ for every $L \in \mathcal{R}$. The *dimension* of P, denoted $\dim(P)$, is the minimum size of a realizer of P. The dimension of P can, alternatively, be defined as the minimum t such that P admits an order preserving embedding into the dominance order on $\mathbb{R}^t$, i.e., elements $x \in X$ have associated vectors $\vec{x} = (x_1, \ldots, x_t)$ with real entries, such that $x \leq_P y$ if and only if $x_i \leq y_i$ for all i, we denote this by $\vec{x} \leq_{\mathsf{dom}} \vec{y}$.

The 1979 edition of Garey and Johnson [4] listed the decision problem, whether a partial order has dimension at most k, in their selection of twelve important problems which were not known to be polynomially solvable or NP-complete. The complexity status was resolved by Yannakakis [9], who used a reduction from 3-colorability to show that the problem is NP-complete for every fixed $k \geq 3$. He also showed that the problem remains NP-complete for every fixed $k \geq 4$ if the partial order is of height 2. The recognition of partial orders of dimension ≤ 2 is easy, it can even be done in linear time [6]. The gap that remained in the complexity landscape was that for partial orders of height 2, the complexity of deciding if the dimension is at most 3 was unknown. In this paper we prove NP-completeness for this case.

Dimension seems to be a particularly hard NP-complete problem. This is indicated by the fact that we have no heuristics or approximation algorithms to produce realizers of partial orders that have reasonable size. Chalermsook et al. [2] show that unless NP = ZPP there exists no polynomial algorithm to approximate the dimension of a partial order with a factor of $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$, where n is the number of elements of the input order.

2 Dimension Three and Triangle Containment

The $t = 3$ case of the following proposition tells us that 3-dimensional orders are the containment orders of homothetic triangles in the plane. The $t = 2$ case is the equivalence between 2-dimensional orders and containment orders of intervals.

Proposition 1 *The dimension of a partial order $P = (X, \leq_P)$ is at most t if and only if P is isomorphic to the containment order of a family of homothetic simplices in $\mathbb{R}^{t-1}$.*

Proof. "⇒" Let $x \to \hat{x}$ be an order preserving embedding of P to $\mathbb{R}^t$. With a point $\hat{x}$ in $\mathbb{R}^t$ associate the orthogonal cone $C(\hat{x}) = \{p \in \mathbb{R}^t \,:\, p \leq_{\mathsf{dom}} \hat{x}\}$. Note that $x \leq_P y$ if and only if $C(\hat{x}) \subseteq C(\hat{y})$. Consider an oriented hyperplane H, such that for all $x \in X$ the point $\hat{x}$ is in the positive halfspace H^+ of H. For $x \in X$, the intersections of the cone $C(\hat{x})$ with H is a $(t-1)$-dimensional polytope $\Delta(x)$ with t vertices, i.e., a simplex. The simplices for different elements are homothetic and $\Delta(x) \subseteq \Delta(y)$ iff $C(\hat{x}) \subseteq C(\hat{y})$ iff $x \leq_P y$. Hence, P is isomorphic to the containment order of the family $\{\Delta(x) \,:\, x \in X\}$ of homothetic simplices in $H \cong \mathbb{R}^{t-1}$.
"⇐" Now suppose that there is a containment order of a family $\mathcal{F} = \{\Delta(x) \colon x \in X\}$ of homothetic simplices in $\mathbb{R}^{t-1}$ that is order isomorphic to P. Apply an affine transformation to get a family $\mathcal{F}' = \{\Delta'(x) : x \in X\}$ of regular simplices in $H = \mathbb{R}^{t-1}$ with the same containment order. Embed H into $\mathbb{R}^t$ with normal vec-

*The full version is available at `arXiv:1501.01147`

[1]Institut für Mathematik, Technische Universität Berlin, D-10623 Berlin, Germany.
[3]Department of Software and Computer Science Education, Charles Univ., Malostranské nám. 25, 11800 Praha 1, CZ.

[a]Partially supported by DFG grant FE-340/7-2 and ESF EuroGIGA.
[b]Supported by Berlin Mathematical School (BMS).
[c]Partially supported by a Czech research grant GAČR GA14-10799S.

tor $\mathbb{1}$. For each $\Delta'(x)$ there is a unique point $\hat{x} \in \mathbb{R}^t$ such that $C(\hat{x}) \cap H = \Delta'(x)$. Since the containment orders of $\{C(\hat{x}) : x \in X\}$ and $\{\Delta'(x) : x \in X\}$ are isomorphic we identify $x \to \hat{x}$ as an order preserving embedding of P into $(\mathbb{R}^t, \leq_{\mathsf{dom}})$. □

2.1 Lemmas for triangle containment

From now on we will restrict the attention to partial orders of height 2. Note that these orders have a bipartite comparability graph. Conversely, any bipartite graph with black and white vertices can be seen a height 2 order, define $u < w$ whenever u is white, w is black and (u, w) is an edge. Hence, partial orders of height 2 and bipartite graphs are essentially the same.

Given a triangle containment representation of a bipartite graph $G = (V, E)$, let $B(V)$ be the set of barycenters of the triangles (it can be assumed that all barycenters are different). Define the β-graph $\beta(G)$ as the straight line drawing of G with vertices placed at their corresponding points of $B(V)$. The following lemma allows some control on the crossings of edges in $\beta(G)$.

Two edges are called *strongly independent* if they share no vertex (i.e., they are independent) and they are the only edges induced on their four vertices.

Lemma 1 (disjoint paths lemma) *In $\beta(G)$, there is no crossing between strongly independent edges.*

The proof of the lemma is very similar to the proof of the easy direction of Schnyder's theorem [8, Thm. 4.1].

For the reduction we construct a bipartite graph with an embedding in the plane which has only few crossings. Most of these crossings between edges occur locally in subgraphs that are named *rotor*. A rotor has a center, which is an adjacent pair u, v of vertices. Additionally there are some non-crossing paths p_i. Each path p_i is connected to the center either with an edge (x_i, u) or with an edge (x_i, v) where x_i is an end-vertex of the path. The interesting case of a rotor is an *alternating rotor*. In this case it is possible to add a simple closed curve γ to the picture such that (1) γ contains the center and (2) there is a collection of six paths intersecting the curve γ cyclically so that paths leading to u and paths leading to v alternate. Figure 1(a) shows a triangle containment representation of an alternating rotor.

In the representation of an alternating rotor there are six *ports* where paths can attach to the center.The ports alternatingly belong to the center vertices u and v. This property is captured by the schematic picture of an alternating rotor given in Figure 1(d).

An *alternating 8-rotor* is a rotor with an 8 alternation, i.e., there are eight paths $p_1, \ldots, p_8$ intersecting a simple closed curve γ that contains the center cyclically in the order of indices such that paths leading to u and paths leading to v alternate.

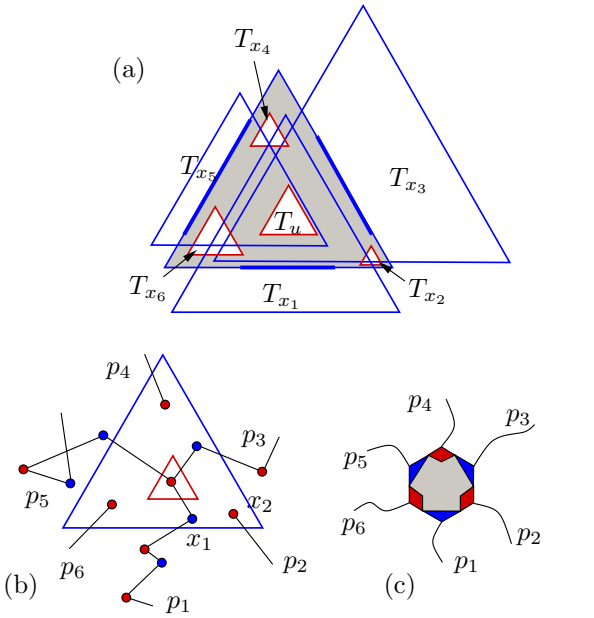


Fig. 1: *Part (a) shows a triangle containment representation of a rotor. Part (b) illustrates how the paths p_1, p_3 and p_5 together with u in the β-graph partition the interior of T_v in three regions, where the paths p_2, p_4 and p_6 start. This implies that the six paths of an alternating rotor have to use all six ports of the center. Part (c) shows a schematized picture of an alternating rotor.*

Lemma 2 *There is no triangle containment representation of an alternating 8-rotor such that in the β-graph of the representation the eight paths $p_1, \ldots, p_8$ intersect a simple closed curve around the center of the rotor in the order of the indices.*

3 The Reduction

3.1 Decision problems

Dimension 3 for Height 2 Orders (3DH2)

Instance: A partial order $P = (X, <)$ of height 2.

Question: Is the dimension of P at most 3?

Bipartite Triangle Containment Representation (BTCon)

Instance: A bipartite graph $G = (V, E)$.

Question: Does G admit a containment representation with a family of homothetic triangles?

From the $t = 3$ case of Proposition 1, it follows that the decision problems 3DH2 and BTCon are polynomially equivalent. In this section we design a reduction from P-3-CON-3-SAT(4) to show that BTCon and, hence, also 3DH2, is NP-complete.

Recall that the *incidence graph* of a SAT instance Φ is a bipartite graph whose vertices are in correspondence to the clauses on one side of the partition, and to the variables of Φ on the other side. Edges of the

incidence graph correspond to membership of a variable in a clause.

P-3-Con-3-SAT(4)

Instance: A SAT instance Φ with the additional properties:

- The incidence graph is planar and 3-connected.
- Each clause consists of 3 literals.
- Each variable contributes to at most 4 clauses.

Question: Does Φ admit a satisfying truth assignment?

The NP-hardness of P-3-CON-3-SAT(4) was shown by Kratochvíl [5]. One of the first applications was to show the hardness of recognizing grid intersection graphs.

The advantage of working with 3-connected planar graphs, instead of just planar graphs, is that in the 3-connected case the planar embedding is unique up to the choice of the outer face. Our reduction is inspired by the reduction from [5], and even more so by the recent NP-completeness proof for the recognition of unit grid intersection graphs with arbitrary girth [7].

3.2 The idea for the reduction

Let Φ be an instance of P-3-CON-3-SAT(4). With Φ we assume a fixed embedding of the incidence graph I_Φ in the plane. Our aim is to construct a bipartite graph G_Φ such that G_Φ has a triangle containment representation if and only if Φ is satisfiable. The construction of G_Φ is done by replacing the constituents of I_Φ by appropriate gadgets. The graph G_Φ will be very sparse and 'almost planar' in the sense that few edge-contractions are sufficient to make the graph planar. (The edges that have to be contracted are concentrated at rotors and gates of the clause gadgets). The planar graph G^*_Φ obtained by contracting these edges is a subdivision of a graph whose plane embedding is essentially unique. Here, essentially unique means unique up to reflections of symmetric subgraphs that correspond to edges of I_Φ. Because of Lemma 1, the essentially unique embedding of G^*_Φ has to be respected by a triangle containment representation of G_Φ.

In the construction of G_Φ, every edge of the incidence graph I_Φ is replaced by a pair of paths of appropriate length. These two paths join a variable gadget and a clause gadget. Such a pair of paths is called an *incidence strand*. The two paths of an incidence strand are two paths of a rotor at their clause end. At this rotor the two paths are incident to different center vertices, hence, they are distinguishable and we may think of one of them as the green path and the other as the yellow path. Assuming a triangle containment representation, we can look at the two paths of an incidence strand in the β-graph. There they can not cross each other and they can not be crossed by any other edge (Lemma 1). Hence if we look at the strip bounded by the two paths with the direction from the variable gadget to the clause gadget, we either see the green path on the left and the yellow path on the right boundary or the other way round. This yields an 'orientation' of the incidence strand. The orientation is used to transmit the truth assignment from the variable to the clause.

The notion of oriented strands is crucial for the design of the clause gadgets and variable gadgets.

3.3 The clause gadget

The clause gadget consists of a rotor surrounded by a cycle and two paths that fix the rotor in the interior of the cycle (magenta). The paths of the three incidence strands which lead to the clause are also connected to the rotor. Three vertices of the cycle have two extra edges connecting to the two paths of an incidence strand. This construction, we call it *gate*, enables the incidence strands to enter the interior of the cycle. Figure 2 shows the clause gadget as a graph and in a more schematic view.

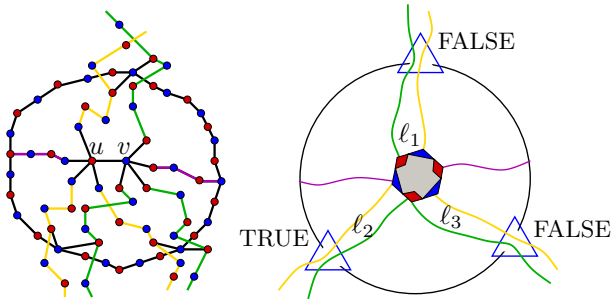


Fig. 2: *Left: The graph of a clause gadget. Right: A schematic view for the case where literals evaluate to* (F, T, F).

The orientation of the incidence strand corresponding to a literal transmits TRUE if one of the two paths of the strand can share a port with one of the magenta paths, and it transmits FALSE if the two paths together with an adjacent magenta path have to use three different ports of the rotor. If all 3 literals evaluate to FALSE we get an alternating 8-rotor, in all other cases we can find a good assignment of the paths to the ports of the rotor.

Proposition 2 *The clause gadget (with sufficiently long paths) has a triangle containment representation if and only if the two paths of the incidence strand of at least one literal are in the orientation representing TRUE.*

3.4 The variable gadget

The variable gadget corresponding to a variable x depends on the number of occurrences of the variable in

clauses. Figure 3 shows the case where the variable has three occurrences.

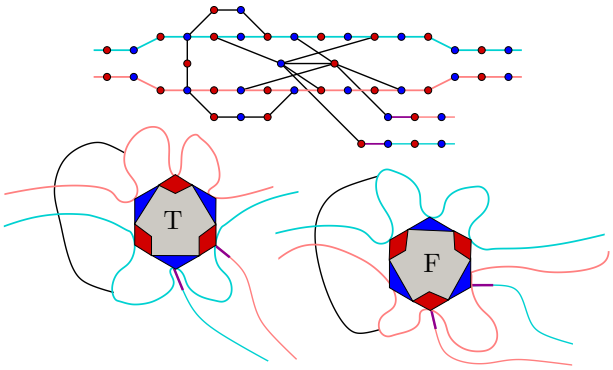


Fig. 3: *The variable gadget for a variable with three occurrences. The upper picture shows the variable gadget as a bipartite graph. The schematic drawings below show the two possible states of the variable gadget which correspond to the two possible values of the variable.*

In the Figure the combined incidence strands of two occurrences are shown in cyan and pink. The adjacencies of vertices on these paths to the center vertices of the rotor between them make an alternating rotor. The duty of the black path is to shield two of the ports of the rotor from outer access. There remain two ports of the rotor where the paths of the third occurrence (magenta ends) can connect to the center of the rotor. Switching the orientation of the first two occurrences also makes the strand of the third occurrence switch orientation, i.e., the truth values transmitted by the three strands are synchronized. Connections between a stands of the variable gadget and the respective strand of a clause gadget have to be adjusted so that the correct truth value is received at the clause. Such an adjustment consists in deciding whether the pairing is (cyan-green,pink-yellow) or (cyan-yellow,pink-green). Which of the pairings has to be chosen depends on (1) which strand of the variable gadget is in question, (2) the position of the literal in the clause, and (3) whether there is a negation involved.

4 Extensions and open questions

In the full version we extend the result by showing that the recognition of containment graphs of points and translates of a fixed (unit) triangle (PUTCon) is NP-complete. Even more, we have the following *sandwich*-result: recognition of every class $\mathcal{C}$ of bipartite graphs such that $\mathcal{C}$ contains all YES instances of PUTCon and $\mathcal{C}$ is contained in the set of all YES instances of BTCon is NP-hard.

From this we also obtain hardness of recognizing bipartite graphs that admit a special type of triangle intersection representation: The two color classes are represented by sets $\Delta_X = \{T_x \ : \ x \in X\}$ and $\Delta_Y = \{T_y^* \ : \ y \in Y\}$ of triangles such that

- Triangles in each of the families are pairwise homothetic while there is a point reflection transforming T_x into T_y^*.
- $(x, y) \in E$ if and only if $T_x \cap T_y^* \neq \emptyset$.
- Within each of the two families Δ_X and Δ_Y there is no containment.

The reduction shows that maximum degree at least 5 is enough to make BTCon hard. From Schnyder's work [8] we know that bipartite graphs where the degree in one of the color classes is 2 are yes instances for BTCon if and only if they are incidence orders of planar graphs. What about maximum degree 3?

- What is the complexity of deciding whether a bipartite graph of maximum degree 3 admits a BTCon representation?

We can also restrict the class of inputs to planar bipartite graphs. It is known that the incidence order of vertices and faces of a 3-connected planar graph is of dimension 4, see [1], moreover there are outerplanar graphs whose incidence order of vertices and faces is of dimension 4, see [3]. Is it hard to decide whether a planar bipartite graph is of dimension 3? Or in terms of triangle containments:

- What is the complexity of deciding whether a planar bipartite graph admits a BTCon representation?

References

[1] G. Brightwell and W. T. Trotter, *The order dimension of convex polytopes*, SIAM J. Discrete Math., 6 (1993), 230–245.

[2] P. Chalermsook, B. Laekhanukit, and D. Nanongkai, *Graph products revisited: Tight approximation hardness of induced matching, poset dimension and more*, in SODA, 2013, 1557–1576.

[3] S. Felsner and J. Nilsson, *On the dimension of outerplanar maps*, Order, 28 (2011), 415–435.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[5] J. Kratochvíl, *A special planar satisfiability problem and a consequence of its NP-completeness*, Discrete Applied Mathematics, 52 (1994), 233–252.

[6] R. M. McConnell and J. Spinrad, *Linear-time transitive orientation*, in SODA, 1997, 19–25.

[7] I. Mustaţă and M. Pergel, *Unit grid intersection graphs: Recognition and properties*, CoRR, abs/1306.1855 (2013).

[8] W. Schnyder, *Planar graphs and poset dimension*, Order, 5 (1989), 323–343.

[9] M. Yannakakis, *The complexity of the partial order dimension problem*, SIAM J. Algebraic Discrete Methods, 3 (1982), 351–358.

Flow Diagrams for Trajectory Analysis

Kevin Buchin* Maike Buchin† Joachim Gudmundsson‡ Michael Horton‡ Stef Sijben†

Abstract

We propose *movement flow diagrams* as a concept to provide a summary for a large number of trajectories and study the problem of computing compact flow diagrams. We show that for a large number of trajectories it is unlikely that efficient algorithms to compute a flow diagram of minimum size exist. More specifically, the problem is $W[1]$-hard if the number of trajectories is taken as a parameter. For a small number of trajectories we present efficient algorithms.

1 Introduction

More and more movement data are recorded in a wide range of applications like sports, traffic analysis and behavioral ecology. In particular, recent advances in tracking technology have led to a large amount of collective movement being recorded. This leads to the question of how to represent these data compactly.

For a single trajectory a common way to obtain a compact representation is *simplification*. Trajectory simplification typically focusses on determining a subset of the data that represents the trajectory well in terms of the location over time [9]. If the focus is less on the location as such but other characteristics of the trajectory then *segmentation* [6] is used to partition a trajectory into a small number of subtrajectories, where each subtrajectory is homogeneous with respect to some characteristic. This allows to represent a trajectory compactly as a sequence of characteristics.

For several trajectories other techniques apply. For a set of similar trajectories one *representative* can be computed [4]. Less similar trajectories may be captured by a *grouping structure* [5]. These approaches again focus on location over time. A large set of trajectories might contain very unrelated trajectories, hence *clustering* may be used. Clustering on complete trajectories will not represent information about interesting parts of trajectories.

We therefore are interested in generalizing the concept of segmentation to sets of trajectories. Consider the trajectories in Fig. 1. Each individual trajectory has segments of directed movement and stops. The flow diagram on the right gives a joint representation of these trajectories. Travel phases are not modelled in the flow diagram, since they are not relevant in this example. If the location of the stops is not relevant we could merge the nodes S_4 and S_5.

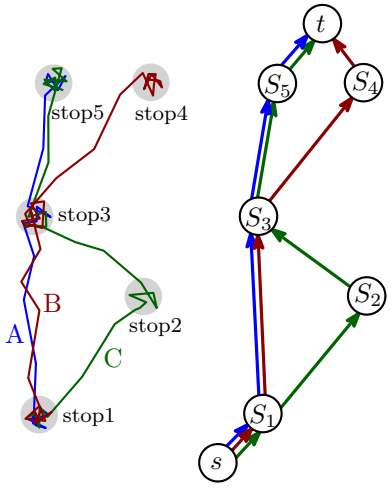


Figure 1: A set of trajectories with some stops and a flow diagram representing the stops.

We study the following problem: Given a group of trajectories, how to give a compact representation of their movement characteristics? For this the trajectories are segmented according to given criteria, and the resulting segmentations are represented in one flow diagram. Given a group of trajectories and a set of criteria, our goal is to find a smallest flow diagram.

After discussing preliminaries and related work, we show in Section 3 that finding a smallest flow diagram is $W[1]$-hard if the number of trajectories is taken as a parameter. In Section 4 we present polynomial-time algorithms for a fixed number of trajectories.

2 Preliminaries

A trajectory T of size n is a sequence of n positions in space and time. A subtrajectory T' of T is a subsequence of T. Let $C_1, \ldots, C_k$ be k criteria, i.e. boolean functions on the set of subtrajectories. These criteria are often geometric in nature. A segmentation of T according to the criteria is a partition into subtrajectories, such that consecutive subtrajectories overlap in exactly one of the n positions and each subtrajec-

*Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands, k.a.buchin@tue.nl

†Dept. of Mathematics, Ruhr-Universität Bochum, Germany, {Maike.Buchin,Stef.Sijben}@rub.de

‡School of Information Technologies, The University of Sydney, Australia, {joachim.gudmundsson, mhor9676}@sydney.edu.au

tory fulfils a criterion. An optimal segmentation is one with a minimal number of subtrajectories.

Now let $\tau_1, \ldots, \tau_m$ be m trajectories of size (at most) n each. We propose to represent a joint segmentation of the trajectories by a *(movement) flow diagram*. A flow diagram is a labelled directed acyclic graph (DAG) where the node labels are criteria except for two nodes labelled s and t respectively. A segmentation (given as a sequence of criteria) is represented in a flow diagram if it appears as a simple path from s to t, where s is concatenated at the beginning and t at the end of the sequence. Our goal is to find a minimal flow diagram that represents at least one valid segmentation for each of a set of m trajectories. As size of the flow diagram, which we aim to minimize, we consider its number of nodes. Note that this problem is "two-fold" in the sense that it asks to find both the segmentations of a group of trajectories, and the flow diagram that represents these.

Note that a criterion might make a statement about just one or about several trajectories. In the first case we can use the corresponding node of the flow diagram to represent segments from different trajectories even if they do not correspond in space or time (e.g., merging stops S_4 and S_5 in Fig. 1). In the second case we may require a correspondence in space and/or time for all segments represented by a certain node.

Criteria-based segmentation for one trajectory has been discussed in a series of papers for different types of criteria: decreasing monotone criteria [6, 7], non-monotone criteria [3], and stable criteria [1]. Here, criteria are distinguished by how often they "change validity". A criterion is *decreasing monotone*: if a segment fulfils the criterion, then so does each subsegment. *Increasing monotone* is defined analogously, and *stable* generalizes both of these concepts. Here, we consider general and decreasing monotone criteria.

The efficiency of segmentation algorithms also depends on the efficiency to check criteria on subtrajectories. Here, we distinguish between *computation* and *update cost*. That is, the time to compute a criterion on a subtrajectory without and with knowing the result on a subtrajectory of size one smaller.

Another important distinction is between *fixed* and *variable parameter* criteria, i.e., criteria which (do not) allow to choose parameters. For instance, location within disc is a variable criterion, and criteria on attribute ranges can be both variable or fixed.

For multiple trajectories we distinguish *dependent* and *independent* criteria. A criterion is independent if checking whether a set of subtrajectories fulfils it can be done for each subtrajectory independently. A similar distinction of computation and update cost, and fixed and variable parameters was made in [7].

In contrast to criteria-based segmentation, a recent paper studies segmentation based on a parameterized movement model [2].

3 Hardness Results

Flow Diagram (FD)
Instance: A set $\mathcal{T}$ of m trajectories, each of length $\leq n$, a set $\mathcal{C}$ of criteria, and a positive integer λ.
Question: Does there exist a flow diagram with $\leq \lambda$ nodes that represents a valid segmentation w.r.t. $\mathcal{C}$ for each trajectory in $\mathcal{T}$?

We show that the problem mentioned above is NP-hard and that the problem is $W[1]$-hard in the number of trajectories. Unless $W[1] = FPT$ this rules out the existence of algorithms with time complexity of $O(f(m) \cdot (nk)^c)$, for some constant c. To show this we describe two reductions: one from Shortest Common Supersequence and one from Set Cover problem.

Reduction from SCS

Shortest Common Supersequence (SCS)
Instance: A set of strings $R = \{r_1, r_2, \ldots, r_k\}$ over an alphabet Σ, a positive integer λ.
Question: Does there exist a string $s \in \Sigma^*$ of length $\leq \lambda$ that is a supersequence of each string in R?

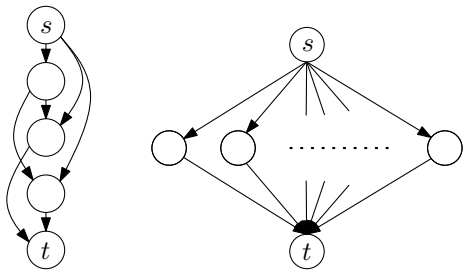
This problem has been extensively studied (see [8] and references therein). In particular, the SCS problem parameterized in the number of strings is $W[1]$-hard even over a constant-size alphabet [11] and the SCS problem over a binary alphabet is NP-hard [12].

Given an instance $I = (R = \{r_1, \ldots, r_m\}, \Sigma, \lambda)$ of SCS construct an instance of FD as follows. Each character c_l in Σ corresponds to a criterion C_l. Each string r_i corresponds to a trajectory τ_i, where $\tau_i[j]$ fulfils $c_{r_i[j]}$ and no other criteria.

An algorithm for FD outputs a flow diagram F of size f. Given F one can compute a linear sequence of the nodes of F using topological sort, as shown in Fig. 2a. The linear sequence has $f - 2$ nodes (omitting the start and end nodes of F) and its node labels are a supersequence of each string in R. It follows that there exists a flow diagram for the instance $(\mathcal{T} = \{\tau_1, \ldots, \tau_m\}, \mathcal{C} = \{C_1, \ldots, C_{|\Sigma|}\})$ of size $\leq \lambda + 2$ if and only if the SCS instance has a solution of size $\leq \lambda$. Note that F contains a linear sequence of nodes (after topological sort), which correspond to a supersequence, and a set of directed edges. Consequently a solution for the FD problem can easily be transformed to a solution for the SCS problem, but not vice versa.

Theorem 1 *The FD problem parameterized in the number of trajectories is $W[1]$-hard even when the number of criteria is constant.*

Theorem 2 *The FD problem is NP-hard even when the number of criteria is two.*



(a) (b)

Figure 2: Examples of flow diagrams produced by the reductions: (a) From Shortest Common Supersequence. (b) From Set Cover.

Reduction from Set Cover

Set Cover (SC)
Instance: A set $E = \{e_1, \ldots, e_m\}$, a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets of E and a positive integer λ.
Question: Does there exist a set of $\leq \lambda$ items in S whose union equals E?

Set Cover is well known to be NP-hard, and also hard to approximate: For any $0 < c < 1/4$, it cannot be approximated within a factor of $c \log m$ in polynomial time unless $NP \subseteq DTIME(m^{\text{polylog}\, m})$ [10].

Given an instance $I = (E = \{e_1, e_2, \ldots, e_m\}, \mathcal{S} = \{S_1, S_2, \ldots, S_n\}, \lambda)$ of Set Cover construct an instance of FD as follows. Each item e_i in E corresponds to a trajectory τ_i of length two. Each subset S_j corresponds to a criterion C_j. If a S_j contains e_i then the whole trajectory τ_i fulfils criterion C_j.

An algorithm for FD given an instance outputs a flow diagram F of size f, as depicted in Fig. 2b. Given F, the labels of its interior nodes correspond to a set of subsets in $\mathcal{S}$ whose union is E. F has $\leq \lambda + 2$ nodes if and only if there are $\leq \lambda$ subsets in $\mathcal{S}$ cover E.

Theorem 3 *The FD problem is NP-hard even when the length of every trajectory is* 2.

Theorem 4 *For any $0 < c < 1/4$, the FD problem cannot be approximated within a factor of $c \log m$ in polynomial time unless $NP \subseteq DTIME(m^{\text{polylog}\, m})$.*

4 Algorithms

In this section, we present dynamic programming algorithms that compute the smallest flow diagram representing a set of m trajectories of length n with respect to a set of k criteria. Let $T(m, n)$ be the cost of testing whether a set of m subtrajectories of length at most n fulfils a criterion. First, we present an algorithm that solves the general case. Then, we present more efficient algorithms for several classes of criteria.

General criteria

We represent all prefixes of the trajectories as vertices in a n^m size regular grid, where the vertex with coordinates $(x_1, x_2, \ldots, x_m)$, $1 \leq x_1, \ldots, x_m \leq n$, represents the trajectory prefixes $(\tau_1[1, x_1], \tau_2[1, x_2], \ldots, \tau_m[1, x_m])$, where $\tau_i[1, x_j]$ denotes the subtrajectory of the first x_j positions. We construct a graph G on these vertices to represent the valid flow diagrams as follows: An edge between two vertices $v = (x_1, \ldots, x_m)$ and $v' = (x'_1, \ldots, x'_m)$, labelled by some criterion c_i, represents adding (at most) one new segment $\tau_j[x_j, x'_j]$ fulfilling c_i for each j. The edge must have $x_j \leq x'_j$ for each $j \in \{1, \ldots, m\}$ and there must be at least one j for which $x_j < x'_j$. Furthermore, for all segments $\tau_j[x_j, x'_j]$ such that $x_j < x'_j$, the segment must fulfil the criterion c_i. If $x_j = x'_j$, then this edge adds no new segment to τ_j. This ensures that the flow diagram represents valid segmentations.

Let v_s be the vertex with coordinates $(1, \ldots, 1)$ and let there be an additional vertex v_t outside the grid. Now, a path in G from v_s to a vertex v represents a valid segmentation of some prefix of each trajectory, and defines a flow diagram that describes these segmentations in the following way: The empty path represents the flow diagram consisting only of the start node s. Every edge of the path adds one new node to the flow diagram, labelled by the criterion that the covered segments fulfil, i.e. the label of the edge in G. Also, the flow diagram gets an edge from every node representing a predecessor of a covered segment, or from s if a covered segment is the first in a segmentation. If $v = v_t$, the target node t is added to the flow diagram, together with its incoming edges.

Lemma 5 *A smallest flow diagram for a set of trajectories is represented by a shortest v_s–v_t path in G.*

G has $n^m + 1$ vertices. There is at most one edge per criterion between a pair of vertices, so G has $O(n^{2m}k)$ edges. For each edge, we need to test whether the represented subtrajectories fulfil the criterion.

Theorem 6 *A smallest flow diagram for m trajectories of length n and k criteria can be computed in $O(n^{2m}k \cdot T(m, n))$ time.*

Decreasing monotone and independent criteria

If all criteria are decreasing monotone and independent, we can avoid constructing the full graph and thus speed up the algorithm.

From a given vertex with coordinates $(x_1, \ldots, x_m)$, we can greedily move as far as possible along the trajectories, since the monotonicity guarantees that this never leads to a solution that is worse than one that generates shorter segments. For a given criterion C_j,

we can compute for each trajectory τ_i independently the maximum x_i' such that $\tau_i[x_i, x_i']$ satisfies C_j. This produces coordinates $(x_1', \ldots, x_m')$ for a new vertex, which is optimal if C_j is selected as the next criterion. By considering all criteria we obtain k new vertices. However, unlike the case with a single trajectory presented in [6], there is not necessarily one vertex that is better than all others (i.e. largest ending position), since the vertices are not totally ordered. Instead, we consider all vertices not dominated by another vertex.

Let V_i be the vertices of G that are reachable from v_s in exactly i steps, and let $M(V) := \{v \in V \mid \text{no vertex } u \in V \text{ dominates } v\}$ be the *maximal vertices* of a vertex set V. Then a shortest v_s–v_t path can be computed by computing $M(V_i)$ for increasing i, until a value of i is found for which $v_t \in M(V_i)$.

Lemma 7 *For each $i \in \{1, \ldots, \ell - 1\}$, every vertex in $M(V_i)$ is reachable in one step from a vertex in $M(V_{i-1})$. Here, ℓ is the distance from v_s to v_t.*

$M(V_i)$ is computed by computing the farthest reachable vertex for each $v \in M(V_{i-1})$ and criterion, thus yielding a set D_i of $O(n^{m-1}k)$ vertices, which contains $M(V_i)$ by Lemma 7. The total size of all D_i $(0 \leq i \leq \ell - 1)$ is $O(kn^m)$ and we can compute all $M(V_i)$ in $O((k+m)n^m)$ time.

Theorem 8 *A smallest flow diagram for m trajectories of length n and k independent and decreasing monotone criteria can be computed in time $O(mnk \cdot T(1,n) + (k+m)n^m)$.*

Decreasing monotone and dependent criteria

We can use the same idea as described in the previous section, but for a given starting vertex v and criterion C, there is not a single vertex v' that dominates all vertices reachable from v using this criterion. Instead, there may be up to $\Theta(n^{m-1})$ maximal reachable vertices from v for criterion C. The total size of all D_i $(0 \leq i \leq \ell - 1)$ is $O(kn^{2m-1})$, leading to this result:

Theorem 9 *A smallest flow diagram for m trajectories of length n and k decreasing monotone criteria can be computed in $O(kn^{2m-1}T(m,n) + mn^m)$ time.*

Fixed criteria

We propose a dynamic programming algorithm that computes the optimal flow diagram for fixed criteria, using a table with n levels representing the time steps, each of size k^m, representing all combinations of criteria a set of m subtrajectories can fulfil. The cell with coordinates $(i, c_1, \ldots, c_m)$ stores the minimal flow diagram of the first i observations of each trajectory, ending with each τ_j in criterion C_{c_j}. To compute an entry at level $i+1$, each state from the previous time step is tested as a predecessor and the one yielding the smallest flow diagram is selected.

Theorem 10 *A smallest flow diagram for m trajectories of length n and k fixed criteria can be computed in $O(k^{2m} \cdot mn)$ time, assuming $T(1,1) = O(1)$.*

References

[1] S.P.A. Alewijnse, K. Buchin, M. Buchin, A. Kölzsch, H. Kruckenberg, and M.A. Westenberg. A framework for trajectory segmentation by stable criteria. In *Proc. 22nd ACM SIGSPATIAL Internat. Conf. Adv. Geographic Information Systems (ACM GIS)*, page To appear, 2014.

[2] S.P.A. Alewijnse, K. Buchin, M. Buchin, S. Sijben, and M.A. Westenberg. Model-based segmentation and classification of trajectories. In *Proc. 30th European Workshop on Computational Geometry (EuroCG)*, 2014.

[3] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories for non-monotone criteria. In *Proc. 24th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1897–1911, 2013.

[4] K. Buchin, M. Buchin, M. Van Kreveld, M. Löffler, R. I Silveira, C. Wenk, and L. Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[5] K. Buchin, M. Buchin, M. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. In *Proc. 13th Internat. Conf. Algorithms and Data Structures*, pages 219–230. Springer-Verlag, 2013.

[6] M. Buchin, A. Driemel, M. van Kreveld, and V. Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3:33–63, 2011.

[7] M. Buchin, H. Kruckenberg, and A. Kölzsch. Segmenting trajectories based on movement states. In *Proc. 15th Internat. Sympos. Spatial Data Handling (SDH)*, pages 15–25. Springer-Verlag, 2012.

[8] C.B. Fraser and R.W. Irving. Approximation algorithms for the shortest common supersequence. *Nordic Journal of Computing*, 2(3):303–325, September 1995.

[9] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. *Computational geometry*, 42(9):825–841, 2009.

[10] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proc. 25th Annual ACM Sympos. Theory of Computing*, STOC '93, pages 286–293, New York, NY, USA, 1993. ACM.

[11] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Computer and System Sciences*, 67(4):757 – 771, 2003.

[12] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2):187 – 198, 1981.

Homotopy Measures for Representative Trajectories*

Erin Chambers† Irina Kostitsyna‡ Maarten Löffler§ Frank Staals§

Abstract

An important task in trajectory analysis is defining a meaningful representative for a set of similar trajectories. How to formally define and find such a representative is a challenging problem. We propose and discuss two possible definitions. In both definitions we use only the geometry of the trajectories, that is, no temporal information is required, and measure the quality of the representative using the homotopy area between the representative and the input trajectories. Computing an optimal representative turns out to be NP-hard for one of the definitions, whereas the other definition allows efficient algorithms for a reasonable class of input trajectories.

1 Introduction

Extracting a meaningful representative trajectory from a collection of similar trajectories is an important open problem in GIS that has recently received considerable attention [1–3, 9–11, 13, 14]. Fig. 1 illustrates our setting, where trajectories are embedded in the plane; the goal is to find a good representative that captures important features shared by most of the input curves.

In [4], Buchin et al. investigate whether a reasonable notion of a median exists that depends only on the intersections in a set of trajectories, while essentially not using the geometry in any way. The authors also incorporate a notion of the topology of the underlying space, by placing obstacles in large open regions and restricting the class of trajectories to the same homotopy type. They conclude that while computation of the median is possible to some extent, some notion of geometry and topology seems necessary to handle practical situations.

In this paper, we include some geometric and topological information in the selection of a representative curve, namely, the area of the faces in the arrangement of trajectories. We use the *homotopy area* measure from Chambers and Wang [7], which measures similarity by the area swept by a minimum homotopy between two input curves. This notion is particularly attractive in our setting as it implicitly penalizes a representative trajectory for missing large regions without making it necessary to artificially place obstacles in the ambient space.

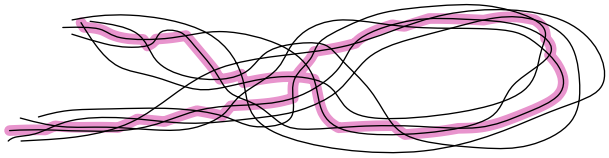


Figure 1: A set of similar trajectories, and the desired representative trajectory.

Clearly, if the trajectories considered are completely dissimilar, and thus have no important common features, there is also no good representative trajectory for them. Therefore, it makes sense to consider only trajectories that have the same general "shape".

Problem Statement. We are given a set of *trajectories* $\mathcal{T} = \{T_1, .., T_n\}$, which for the purposes of this paper are simply curves in the plane. We wish to compute a single trajectory $\mathfrak{m}^*$ that best represents all trajectories in $\mathcal{T}$. As we will use homotopy area to measure the quality of $\mathfrak{m}^*$ we require that: (*i*) all trajectories start and end at the same points, (*ii*) each individual trajectory T_i is *simple*, that is, it has no self-intersections (or else homotopy area is not well defined), and (*iii*) the start and end points lie in the outer face of the arrangement of trajectories.

We require that $\mathfrak{m}^*$ also goes from the common start point to the common end point and has the following properties: (a) $\mathfrak{m}^*$ should consist of segments of the input trajectories, (b) $\mathfrak{m}^*$ should be simple, (c) $\mathfrak{m}^*$ should use each segment in the correct direction (i.e., the same direction as used in the input trajectory), and (d) all segments of a trajectory that appear on $\mathfrak{m}^*$ should appear in the correct order.

Among all possible output trajectories that satisfy these requirements, we wish to select one that represents $\mathcal{T}$ best. We measure this by the distance between the (candidate) median $\mathfrak{m}$ and the trajectories in $\mathcal{T}$. Let $d(\mathfrak{m}, T)$ be the homotopy area between $\mathfrak{m}$ and a trajectory $T \in \mathcal{T}$. We con-

*E.C. is supported by the National Science Foundation under grant CCF-1054779 and IIS-1319573. I.K., M.L., and F.S. are supported by the Netherlands Organisation for Scientific Research (NWO) under grant 612.001.106, 639.021.123, and 612.001.022 respectively.

†Department of Math and Computer Science , Saint Louis University; echambe5@slu.edu.

‡Department of Mathematics and Computer Science, TU Eindhoven; i.kostitsyna@tue.nl.

§Department of Computing and Information Sciences, Utrecht University; [m.loffler,f.staals]@uu.nl.

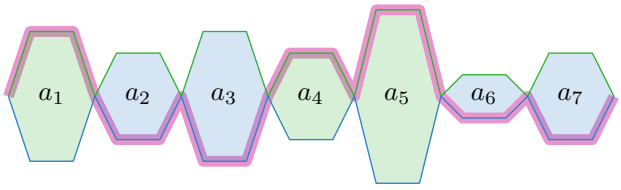


Figure 2: An illustration of the NP-hardness reduction from Partition. The purple curve represents the partition $B = \{a_2, a_3, a_6, a_7\}$ and $G = \{a_1, a_4, a_5\}$.

sider two variants: minimizing the maximum distance $\mathscr{M}(\mathfrak{m}, \mathcal{T}) = \max_{T \in \mathcal{T}} d(\mathfrak{m}, T)$ between $\mathfrak{m}$ and the trajectories in $\mathcal{T}$, and the sum of the distances $\mathscr{D}(\mathfrak{m}, \mathcal{T}) = \sum_{T \in T} d(\mathfrak{m}, T)$ between $\mathfrak{m}$ and the trajectories in $\mathcal{T}$. If $\mathcal{T}$ is clear from the context we will write $\mathscr{M}(\mathfrak{m}) = \mathscr{M}(\mathfrak{m}, \mathcal{T})$ and $\mathscr{D}(\mathfrak{m}) = \mathscr{D}(\mathfrak{m}, \mathcal{T})$.

Results. We show that the first variant considered, minimizing the maximum distance, is NP-hard, even if the trajectories are all x-monotone. For the second variant we show that if the trajectories have a similar "shape" then we can compute a representative minimizing $\mathscr{D}$ efficiently. Quite surprisingly, our results show that under reasonable constraints, the simple median from Buchin et al. [4] that does not incorporate areas in any way, remains the optimal choice for minimizing $\mathscr{D}$.

2 Minimizing the Maximum Distance $\mathscr{M}$

We first show that the problem of minimizing the maximum distance between the median trajectory and all other trajectories in NP-hard, even for the case of a constant number of x-monotone input curves. Our reduction proceeds from the Partition problem, which, given a set $A = \{a_1, .., a_n\}$ of positive integers, asks if there is a partition of A into sets B and G such that $\sum(B) = \sum(G) = \sum(A)/2$, where $\sum(X) = \sum_{a \in X} a$.

Given the set A, we construct two x-monotone trajectories (curves) T_B and T_G such that the faces between successive intersections have area equal to some $a_i \in A$. See Fig. 2 for an illustration.

Any candidate trajectory $\mathfrak{m}$ corresponds to a partition of A into B and G: $a_i \in B$ if and only if $\mathfrak{m}$ uses the edges of T_B that bound the face corresponding to a_i. It follows that the homotopy area between $\mathfrak{m}$ and T_B is exactly $\sum(B)$. Similarly, the homotopy area between $\mathfrak{m}$ and T_G is $\sum(G)$, and thus $\mathscr{M}(\mathfrak{m}) = \max\{\sum(B), \sum(G)\}$. Let $\mathfrak{m}^*$ be a trajectory minimizing $\mathscr{M}$. We have that $\mathscr{M}(\mathfrak{m}^*) = \sum(A)/2$ if and only if A can be partitioned such that $\sum(B) = \sum(G)$. It follows that minimizing $\mathscr{M}$ is (weakly) NP-hard.

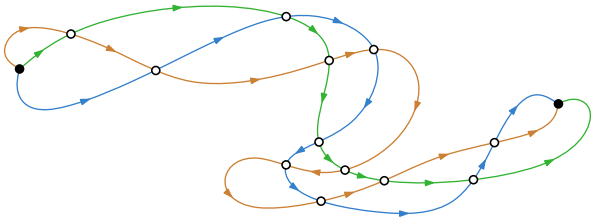


Figure 3: The trajectory graph Γ. In this example, Γ is acyclic.

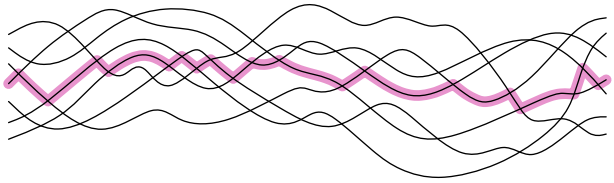


Figure 4: The simple median for a set of x-monotone trajectories.

3 Minimizing the Sum of Distances $\mathscr{D}$

We now describe how to compute a representative that minimizes $\mathscr{D}$ for a set of trajectories $\mathcal{T}$ whose shape is similar. As a warmup, we consider the case in which the trajectories in $\mathcal{T}$ are x-monotone. We then show that this approach extends to the case where the *trajectory graph* Γ is an arbitrary acyclic graph. The set of vertices of Γ consists of the start point, the end point, and the intersection points of the trajectories. An edge in Γ then corresponds to a piece of a trajectory, directed along that trajectory. See Fig. 3.

3.1 x-Monotone Trajectories

In this section we will show that for x-monotone trajectories, the simple median, as defined by Buchin et al. [4], also minimizes the sum of the homotopy areas $\mathscr{D}$. At the starting point s, the simple median starts at the $n/2^{\text{th}}$ curve (ranking the trajectories by their y-coordinate just after s). It switches at every intersection point it encounters, thus staying on the $n/2^{\text{th}}$ trajectory. See Fig. 4. Throughout this section we will consider the trajectories as functions mapping time, represented by the x-axis, to $\mathbb{R}^1$, represented by the y-axis. Thus we use $T(t) = T(x)$ to denote the y-coordinate of the trajectory T at time $t = x$.

To show that the simple median $\mathfrak{m}^*$ minimizes $\mathscr{D}$ we write $\mathscr{D}(\mathfrak{m})$ as an integral $\int f(t)\,\mathrm{d}t$ over time t. At any individual time t, $f(t)$ represents the sum of the lengths of a set of intervals along a vertical line. All intervals share a common endpoint (the value of $\mathfrak{m}$ at time t). The total length of these intervals is minimal when $\mathfrak{m}$ has the same number of trajectories above and below it, that is, when it is the simple median at time t.

Lemma 1 *The simple median minimizes*

$$F(\mathfrak{m}) = \int_t \sum_{T\in\mathcal{T}} |\mathfrak{m}(t) - T(t)| \,\mathrm{d}t.$$

Proof. Let $y_1, .., y_k$ denote the intersection points of the trajectories with a vertical line ℓ_x with $x = t$. Any valid trajectory uses one of the points t_i at time t. We now show that the point y_h, with $h = \lceil n/2 \rceil$, minimizes $F'(y) = \sum_{T\in\mathcal{T}} |y - T(x)|$ at $x = t$. Since the simple median $\mathfrak{m}^*$ is on the h^{th} trajectory at any time t, it thus follows that $\mathfrak{m}^*$ minimizes F.

Assume by contradiction that the point that minimizes F' at x is on t_{i+1}, with $i > h$. The case $i < h$ is symmetric. We have $F'(t_{i+1}) = \sum_{j>i+1}(t_j - t_{i+1}) + \sum_{j<i+1}(t_{i+1} - t_j)$, and $F'(t_i) = F'(i+1) + (n - i)(t_{i+1} - t_i) - i(t_{i+1} - t_i)$. Since $i > h = \lceil n/2 \rceil$ it follows that $F'(i) < F'(i+1)$. Contradiction. □

Given a point p let $\omega(p, \gamma)$ denote the winding number of p with respect to a closed curve γ.

Lemma 2 *Let $\mathcal{T}$ be a set of x-monotone trajectories. The simple median minimizes $\mathscr{D}$.*

Proof. Let $\mathfrak{m}$ be a candidate median trajectory, and let γ_T be the closed curve obtained by concatenating $\mathfrak{m}$ and the reverse of T. All trajectories are x-monotone and have consistent winding numbers, so for any point p any winding number $\omega(p, \gamma_T)$ is either zero, plus one, or minus one. We then apply Lemma 4.3 of Chambers et al. [7] and obtain

$$\begin{aligned}\mathscr{D}(\mathfrak{m}) &= \sum_{T\in\mathcal{T}} d(\mathfrak{m}, T) = \sum_{T\in\mathcal{T}} \left| \int_{p\in\mathbb{R}^2} \omega(p, \gamma_T) \,\mathrm{d}p \right| \\ &= \sum_{T\in\mathcal{T}} \int_{x\in\mathbb{R}} \int_{y\in\mathbb{R}} |\omega((x, y), \gamma_T)| \,\mathrm{d}y \,\mathrm{d}x \\ &= \int_{x\in\mathbb{R}} \sum_{T\in\mathcal{T}} \int_{y\in\mathbb{R}} |\omega((x, y), \gamma_T)| \,\mathrm{d}y \,\mathrm{d}x.\end{aligned}$$

A vertical line ℓ_x with x-coordinate x intersects (the faces of) Γ in a set of intervals $\mathcal{I}(x) = I_1, .., I_k$. All points (values) in an interval I_i have the same winding number $\omega(I_i, \gamma_T)$ with respect to a curve γ_T. So,

$$\begin{aligned}\mathscr{D}(\mathfrak{m}) &= \int_{x\in\mathbb{R}} \sum_{T\in\mathcal{T}} \sum_{I\in\mathcal{I}(x)} \int_{y\in I} |\omega((x, y), \gamma_T)| \,\mathrm{d}y \,\mathrm{d}x \\ &= \int_{x\in\mathbb{R}} \sum_{T\in\mathcal{T}} \sum_{I\in\mathcal{I}(x)} |\omega(I, \gamma_T)| \cdot |I| \,\mathrm{d}x.\end{aligned}$$

Since the trajectories are x-monotone, each trajectory $T \in \mathcal{T}$ intersects a vertical line ℓ_x in exactly one point, namely $(x, T(x))$. Let $J_T \subseteq \ell_x$ be the interval bounded by $\mathfrak{m}(x)$ and $T(x)$. It follows that $|\omega(I, \gamma)| = 1$ if $I \subseteq J_T$ and zero otherwise, and thus

$$\mathscr{D}(\mathfrak{m}) = \int_{x\in\mathbb{R}} \sum_{T\in\mathcal{T}} |\mathfrak{m}(x) - T(x)| \,\mathrm{d}x = F(\mathfrak{m}).$$

The lemma now follows from Lemma 1. □

From Lemma 2 it follows that we can compute a median trajectory using the algorithm of Buchin et al. [4] in $O((N + k)\alpha(N) \log(N))$ time, where N is the total complexity of the input trajectories, and k is the output complexity (which is at most $O(N^2)$).

3.2 Extending to Arbitrary Acyclic Γ

The arguments from the previous section can be extended to the case where the trajectory graph is acyclic. We observed that the function F in Lemma 1 is an integral over time, summing the lengths of intervals along a vertical line. It is easy to see that this generalizes to a sum of intervals along an arbitrary time-varying curve. More formally, let $\ell : [0, 1]\times[0, 1] \to \mathbb{R}^2$ be a continuous map such that at any time t, the curve $\ell(t) = \bigcup_{z\in[0,1]} \ell(t, z)$ intersects each trajectory from $\mathcal{T}$ exactly once. The curve $\mathfrak{m}_\ell$ that at any time t corresponds to the $n/2^{\text{th}}$ intersection point on $\ell(t)$ minimizes the function

$$G(\mathfrak{m}) = \int_{t\in[0,1]} \sum_{T\in\mathcal{T}} len\big(\ell(t), \mathfrak{m}(t), T(t)\big) \,\mathrm{d}t,$$

where $len(C, p, q)$ denotes the length along curve C from p to q. Additionally, we observe that for any two maps ℓ_1 and ℓ_2, these median curves $\mathfrak{m}_{\ell_1}$ and $\mathfrak{m}_{\ell_2}$ are the same curve: the simple median $\mathfrak{m}^*$. To see this, consider sweeping ℓ_1 and ℓ_2 over Γ, while maintaining the edges e_1 and e_2 of Γ currently containing $\mathfrak{m}_{\ell_1}$ and $\mathfrak{m}_{\ell_2}$, respectively. Initially, the $\mathfrak{m}_{\ell_1}$ and $\mathfrak{m}_{\ell_2}$ use the same outgoing edge of s, hence $e_1 = e_2$. The key insight is now that e_i changes only if ℓ_i sweeps over the end point v of e_i. Conversely, if ℓ_i sweeps over a different vertex of Γ, the number of curves intersecting ℓ_i before and after $\mathfrak{m}_{\ell_i}$ does not change. This implies that $\mathfrak{m}_{\ell_1}$ and $\mathfrak{m}_{\ell_2}$ both use the same outgoing edge of v. Repeating this argument gives us that $\mathfrak{m}_{\ell_1} = \mathfrak{m}_{\ell_2} = \mathfrak{m}^*$. Therefore, we conclude:

Lemma 3 *The simple median minimizes $G(\mathfrak{m})$.*

Analogous to Lemma 2 we now rewrite $\mathscr{D}(\mathfrak{m})$ as an integral over time. However, instead of directly mapping time t to a vertical line $x = t$, we map every time t to a curve $\ell(t)$ that intersects each trajectory exactly once. Such a (continuous) map exists, since the trajectories do not contain self intersections, and Γ is acyclic.

It then again follows that all winding numbers are zero, one, or minus one, and thus we can obtain $\mathscr{D}(\mathfrak{m}) = G(\mathfrak{m})$. The following result then follows from Lemma 3.

Lemma 4 *Let $\mathcal{T}$ be a set of trajectories for which Γ is acyclic. The simple median minimizes $\mathscr{D}$.*

Thus, we can again compute a median trajectory using the algorithm of Buchin et al. [4]. Hence:

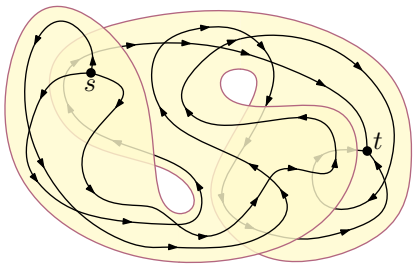


Figure 5: A corridor.

Theorem 5 *Let $\mathcal{T}$ be a set of trajectories with total complexity N and for which Γ is acyclic. A trajectory that minimizes $\mathscr{D}$ can be computed in $O((N+k)\alpha(N)\log(N))$ time, where k is the complexity of the resulting trajectory.*

4 Future Work

Throughout this paper, we have assumed that the trajectories are constrained to be similar, so that the underlying intersection graph is x-monotone or acyclic. When the trajectory graph is not acyclic, we propose a two phase approach. In the first phase we compute a *corridor*, capturing the global shape of the trajectories. In the second phase we compute a concrete curve representing the trajectories in the corridor. The conceptual existance of a corridor is justified by the assumption that the input trajectories are similar. We can formalize the notion of a corridor as follows.

Let $\mathcal{M}$ be a simply connected continuous topological space, let $f : \mathcal{M} \to \mathbb{R}^2$ be a continuous function mapping $\mathcal{M}$ onto $\mathbb{R}^2$, and let $\mathcal{T}_\mathcal{M} = f^{-1}(\mathcal{T})$ denote the set of trajectories lifted onto $\mathcal{M}$ using f^{-1}. We define a corridor to be such a pair $(\mathcal{M}, f)$ for which the trajectory graph $\Gamma_\mathcal{M}$ of the trajectories $\mathcal{T}_\mathcal{M}$ is acyclic. Fig. 5 shows an example.

We can find a representative for the set of trajectories $\mathcal{T}_\mathcal{M}$ using Theorem 5. What remains is to find and compute a corridor $(\mathcal{M}, f)$. We plan to investigate computing a corridor in future work. We will consider lifting the trajectories to a covering space where the underlying graph is acyclic. As long as this lifting to a new space is locally consistent and preserves the fact that any two of the trajectories are homotopic, this allows lifting the homotopy area measure. However, as there are many possible ways to lift, further investigation of the trade-offs involved is necessary.

We focused on using the homotopy area to measure the distance between the trajectories. There are many other alternative measures that take topology and geometry into account. Homotopy width (or homotopic Fréchet distance) [8] and homotopy height [5, 12] are obvious options, as is homology area [6], although it is unclear if any of these are tractable or useful in practice.

References

[1] P. K. Agarwal, M. de Berg, J. Gao, L. J. Guibas, and S. Har-Peled. Staying in the middle: Exact and approximate medians in $\mathbb{R}^1$ and $\mathbb{R}^2$ for moving points. In *CCCG*, pages 43–46, 2005.

[2] R. Basu, B. Bhattacharya, and T. Talukdar. The projection median of a set of points in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 47(2):329–346, 2012.

[3] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *IJCGA*, 21(03):253–282, 2011.

[4] K. Buchin, M. Buchin, M. Kreveld, M. Löffler, R. Silveira, C. Wenk, and L. Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[5] E. W. Chambers and D. Letscher. On the height of a homotopy. In *CCCG*, pages 103–106, 2009.

[6] E. W. Chambers and M. Vejdemo-Johansson. Computing minimum area homologies. *Computer Graphics Forum*, pages n/a–n/a, 2014.

[7] E. W. Chambers and Y. Wang. Measuring similarity between curves on 2-manifolds via homotopy area. In *Proc. 29th Ann. Symp. on CG*, pages 425–434. ACM, 2013.

[8] E. W. Chambers, ric Colin de Verdire, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic frchet distance between curves or, walking your dog in the woods in polynomial time. *CG*, 43(3):295 – 311, 2010.

[9] S. Durocher and D. Kirkpatrick. The projection median of a set of points. *CG*, 42(5):364 – 375, 2009.

[10] S. Gaffney, A. Robertson, P. Smyth, S. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. *Climate Dynamics*, 29(4):423–440, 2007.

[11] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proc. 5th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 63–72, 1999.

[12] S. Har-Peled, A. Nayyeri, M. Salavatipour, and A. Sidiropoulos. How to walk your dog in the mountains with no magic leash. In *Proc. 28th Ann. Symp. on CG*, pages 121–130. ACM, 2012.

[13] J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 593–604, 2007.

[14] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. Data Engin.*, pages 673–684, 2002.

Central Trajectories

Marc van Kreveld* Maarten Löffler* Frank Staals*

Abstract

We study the problem of computing a suitable representative of a set of similar trajectories. To this end we define a *central trajectory* $\mathcal{C}$, which consists of pieces of the input trajectories, switches from one entity to another only if they are within a small distance of each other, and such that at any time t, the point $\mathcal{C}(t)$ is as central as possible. We measure centrality in terms of the radius of the smallest disk centered at $\mathcal{C}(t)$ enclosing all entities at time t, and discuss how the techniques can be adapted to other measures of centrality. For entities moving in $\mathbb{R}^1$ we show that an optimal central trajectory $\mathcal{C}$ representing n trajectories, each consisting of τ edges, has complexity $\Theta(\tau n^2)$ and can be computed in $O(\tau n^2 \log n)$ time. For entities moving in $\mathbb{R}^d$ with $d \geq 2$, the complexity of $\mathcal{C}$ is at most $O(\tau n^{5/2})$ and can be computed in $O(\tau n^3)$ time.

1 Introduction

A *trajectory* is a sequence of time-stamped locations in the plane, or more generally in $\mathbb{R}^d$. Trajectory data is obtained by tracking the movements of e.g. animals [1, 4, 6], hurricanes [8], traffic [7], or other moving entities [5] over time. Large amounts of such data have recently been collected in a variety of research fields. As a result, there is a great demand for tools and techniques to analyze trajectory data.

We study representing a set of (similar) trajectories by a single *representative* trajectory that captures the defining features of all trajectories in the set. Representative trajectories are useful for example in clustering. When choosing a representative trajectory for a group of similar trajectories, the first obvious choice would be to pick one of the trajectories in the group. However, one can argue that no single element in a group may be a good representative, e.g. because each individual trajectory has some prominent feature that is not shared by the rest (see Fig. 1(a)), or no trajectory is sufficiently in the middle all the time. On the other hand, it is desirable to output a trajectory that does consist of *pieces* of input trajectories, because otherwise the representative trajectory may display behaviour that is not present in the input, e.g. because of contextual information that is not available to the algorithm (see Fig. 1(b)).

*Department of Information and Computing Sciences, Universiteit Utrecht, The Netherlands, {m.j.vankreveld|m.loffler|f.staals}@uu.nl. M.L. and F.S. are supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123 and 612.001.022, respectively.

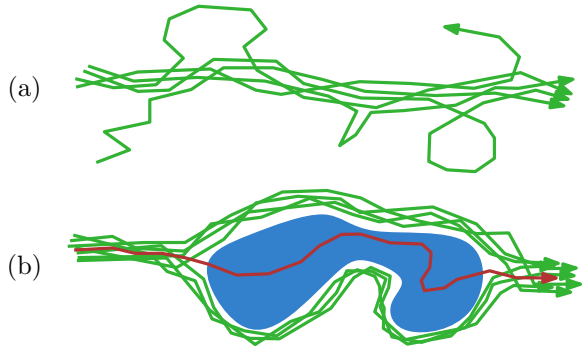


Figure 1: (a) Every trajectory has a peculiarity that is not representative for the set. (b) Taking the pointwise average of a set of trajectories may result in one that ignores context.

Central trajectories. Buchin et al. [2] consider the problem of computing a *median* trajectory for a set of trajectories without time information. Their method considers the trajectories as curves in the plane, and produces a trajectory (curve) that consists of pieces of the input. In this work, we focus on incorporating time into the representative. Ideally, we would output a trajectory $\mathcal{C}$ such that at any time t, $\mathcal{C}(t)$ is the point (entity) that is closest to its farthest entity. Unfortunately, when the entities move in $\mathbb{R}^d$ for $d > 1$, this may cause discontinuities. Such discontinuities are unavoidable: if we insist that the output trajectory consists of pieces of input trajectories *and* is continuous, then in general, there will be no opportunities to switch from one trajectory to another, and we are effectively choosing one of the input trajectories again. At the same time, we do not want to output a trajectory with arbitrarily large discontinuities. An acceptable compromise is to allow discontinuities, or *jumps*, but only over small distances, controlled by a parameter ε. We note that this problem of discontinuities also shows up for representatives without time information and entities moving in $\mathbb{R}^d$, with $d \geq 3$, because the traversed curves generally do not intersect.

Problem description. We are given a set $\mathcal{X}$ of n entities, each moving along a piecewise linear trajectory in $\mathbb{R}^d$ consisting of τ edges. We assume that all trajectories have their vertices at the same times $t_0, .., t_\tau$. For an entity σ, let $\sigma(t)$ denote the position of σ at time t. With slight abuse of notation we will write σ for both entity σ and its trajectory. At a given time t, we denote the distance from σ to the entity farthest away from σ by $D_\sigma(t) = D(\sigma, t) = \max_{\psi \in \mathcal{X}} \|\sigma(t)\psi(t)\|$, where $\|pq\|$ denotes the Euclidean distance between points p and q in $\mathbb{R}^d$.

For ease of exposition, we assume that the trajectories are in general position: that is, no three trajectories intersect in the same point, and no two pairs of entities are at distance ε from each other at the same time.

A *trajectoid* is a function that maps time to the set of entities $\mathcal{X}$, with the restriction that at discontinuities the distance between the entities involved is at most ε. Intuitively, a trajectoid corresponds to a concatenation of pieces of the input trajectories in such a way that two consecutive pieces match up in time, and the end point of the former piece is within distance ε from the start point of the latter piece. More formally, for a trajectoid $\mathcal{T}$ we have that

- at any time t, $\mathcal{T}(t) = \sigma$ for some $\sigma \in \mathcal{X}$, and
- at every time t where $\mathcal{T}$ has a discontinuity, that is, $\mathcal{T}$ *jumps* from entity σ to entity ψ, we have that $\|\sigma(t)\psi(t)\| \leq \varepsilon$.

Note that this definition still allows for a series of jumps within an arbitrarily short time interval $[t, t+\delta]$, essentially simulating a jump over distances larger than ε. To make the formulation cleaner, we slightly weaken the second condition, and allow a trajectoid to have discontinuities with a distance larger than ε, provided that such a large jump can be realized by a sequence of small jumps, each of distance at most ε. When it is clear from the context, we will write $\mathcal{T}(t)$ instead of $\mathcal{T}(t)(t)$ to mean the location of entity $\mathcal{T}(t)$ at time t. We now wish to compute a trajectoid $\mathcal{C}$ that minimizes the function

$$\mathcal{D}(\mathcal{T}) = \int_{t_0}^{t_\tau} D(\mathcal{T}, t)\,\mathrm{d}t.$$

So, at any time t, all entities lie in a disk of radius $D(\mathcal{C}, t)$ centered at $\mathcal{C}(t)$.

Results. Because space restrictions, we present only the situation where entities move in $\mathbb{R}^1$. Our approach can be extended to $\mathbb{R}^d$, as well as other measures of centrality. For these results and all omitted proofs we refer to the full version of this paper [9]. We show that the worst case complexity of a central trajectory in $\mathbb{R}^1$ is $\Theta(\tau n^2)$, and that we can compute one in $O(\tau n^2 \log n)$ time. For entities moving in $\mathbb{R}^d$, for any constant d, the maximal complexity of a central trajectory $\mathcal{C}$ is $O(\tau n^{5/2})$. In this case, computing $\mathcal{C}$ takes $O(\tau n^3)$ time and requires $O(\tau n^2 \log n)$ space.

2 Preliminaries

Let $\mathcal{X}$ be the set of entities moving in $\mathbb{R}^1$. The trajectories of these entities can be seen as polylines in $\mathbb{R}^2$: we associate time with the horizontal axis, and $\mathbb{R}^1$ with the vertical axis (see Fig. 2). We observe that the distance between two points p and q in $\mathbb{R}^1$ is simply their absolute difference, that is, $\|pq\| = |p - q|$.

Let $\mathcal{I}$ be the *ideal* trajectory, that is, the trajectory that minimizes $\mathcal{D}$ but is not restricted to lie on the input trajectories. It follows that at any time t, $\mathcal{I}(t)$ is simply the average of the highest entity $\mathcal{U}(t)$ and the lowest entity $\mathcal{L}(t)$. We further subdivide each time interval $J_i = [t_i, t_{i+1}]$ into *elementary intervals*, such that $\mathcal{I}$ is a single line segment inside each elementary interval.

Lemma 1 *The total number of elementary intervals is* $\tau(n+2)$.

We assume without loss of generality that within each elementary interval $\mathcal{I}$ coincides with the x-axis. To simplify the description of the proofs and algorithms, we also assume that the entities never move parallel to the ideal trajectory, that is, there are no horizontal edges.

Lemma 2 *$\mathcal{C}$ is a central trajectory in $\mathbb{R}^1$ if and only if it minimizes the function*

$$\mathcal{D}'(\mathcal{T}) = \int_{t_0}^{t_\tau} |\mathcal{T}(t)|\,\mathrm{d}t.$$

By Lemma 2 a central trajectory $\mathcal{C}$ is a trajectoid that minimizes the area $\mathcal{D}'(\mathcal{T})$ between $\mathcal{T}$ and the ideal trajectory $\mathcal{I}$. Hence, we can focus on finding a trajectoid that minimizes $\mathcal{D}'$.

3 Complexity of a Central Trajectory

Lemma 3 *For a set of n trajectories in $\mathbb{R}^1$, each with vertices at times $t_0, .., t_\tau$, a central trajectory $\mathcal{C}$ may have worst case complexity $\Omega(\tau n^2)$.*

Two entities σ and ψ are *ε-connected* at time t if there is a sequence $\sigma = \sigma_0, .., \sigma_k = \psi$ of entities such that for all i, σ_i and σ_{i+1} are within distance ε of each other at time t. A subset $\mathcal{X}' \subseteq \mathcal{X}$ of entities is ε-connected at time t if all entities in $\mathcal{X}'$ are pairwise ε-connected at time t. The set $\mathcal{X}'$ is ε-connected during an interval I, if they are ε-connected at any time $t \in I$. We now observe:

Observation 1 *$\mathcal{C}$ can jump from entity σ to ψ at time t if and only if σ and ψ are ε-connected at time t.*

At any time t, we can partition $\mathcal{X}$ into maximal sets of ε-connected entities. The central trajectory $\mathcal{C}$ must be in one of such maximal sets $\mathcal{X}'$: it uses the trajectory of an entity $\sigma \in \mathcal{X}'$ (at time t), if and only if σ is the entity from $\mathcal{X}'$ closest to $\mathcal{I}$. More formally, let $f_\sigma(t) = |\sigma(t)|$, and let $\mathcal{L}(\mathcal{F}) = \min_{f \in \mathcal{F}} f$ denote the lower envelope of a set of functions $\mathcal{F}$.

Observation 2 *Let $\mathcal{X}' \ni \sigma$ be a maximal set of entities that is ε-connected during interval J, and assume that $\mathcal{C} \in \mathcal{X}'$ during J. For any time $t \in J$, we have that $\mathcal{C}(t) = \sigma(t)$ if and only if f_σ is on the lower envelope of the set $\mathcal{F}' = \{f_\psi \mid \psi \in \mathcal{X}'\}$ at time t, that is, $f_\sigma(t) = \mathcal{L}(\mathcal{F})(t)$.*

Let $\mathcal{X}_1, .., \mathcal{X}_m$, denote a collection of maximal sets of entities that are ε-connected during time intervals $J_1, .., J_m$, respectively. Let $\mathcal{F}_i = \{f_\sigma \mid \sigma \in \mathcal{X}_i\}$, and let $\mathcal{L}_i$ be the lower envelope $\mathcal{L}(\mathcal{F}_i)$ of $\mathcal{F}_i$ restricted to interval J_i. A

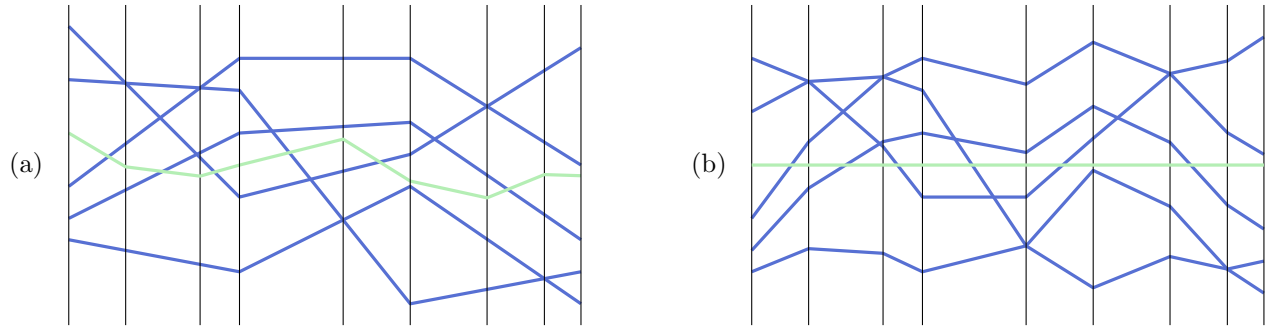


Figure 2: (a) A set of trajectories and the ideal trajectory I. The breakpoints in the ideal trajectory partition time into $O(n\tau)$ intervals. (b) The trajectories after transforming I into a horizontal line.

lower envelope $\mathcal{L}_i$ has a break point at time t if $f_\sigma(t) = f_\psi(t)$, for $\sigma, \psi \in \mathcal{X}_i$. There are two types of break points: (i) $\sigma(t) = \psi(t)$, or (ii) $\sigma(t) = -\psi(t)$. At events of type (i) the modified trajectories of σ and ψ intersect. At events of the type (ii), σ and ψ are equally far from I, but on different sides of I. Let $B = \{(t, \sigma, \psi) \mid \mathcal{L}_i(t) = f_\sigma(t) = f_\psi(t) \wedge i \in \{1, .., m\}\}$ denote the collection of break points from all lower envelopes $\mathcal{L}_1, .., \mathcal{L}_m$.

Lemma 4 *Consider a triplet $(t, \sigma, \psi) \in B$. There is at most one lower envelope $\mathcal{L}_i$ such that t is a break point in $\mathcal{L}_i$.*

Proof. Assume by contradiction that t is a break point in both $\mathcal{L}_i$ and $\mathcal{L}_j$. At any time t, an entity can be in at most one maximal set $\mathcal{X}_\ell$. So if $\mathcal{X}_i$ and $\mathcal{X}_j$ share either entity σ or ψ, then the intervals J_i and J_j are disjoint. It follows t cannot lie in both intervals, and thus cannot be a break point in both $\mathcal{L}_i$ and $\mathcal{L}_j$. Contradiction. □

Lemma 5 *Let $\mathcal{A}$ be an arrangement of n lines, describing the movement of n entities during an elementary interval J. If there is a break point $(t, \sigma, \psi) \in B$, with $t \in J$, of type (ii), then $\sigma(t)$ and $\psi(t)$ lie on the boundary $\partial\mathcal{Z}$ of the zone $\mathcal{Z}$ of I in $\mathcal{A}$.*

Lemma 6 *Let $\mathcal{A}$ be an arrangement of n lines, describing the movement of n entities during an elementary interval J. The total number of break points $(t, \sigma, \phi) \in B$, with $t \in J$, of type (ii) is at most $6.5n$.*

Lemma 7 *The total complexity of all lower envelopes $\mathcal{L}_1, .., \mathcal{L}_m$ on $[t_i, t_{i+1}]$ is $O(n^2)$.*

Theorem 8 *Given a set of n trajectories in $\mathbb{R}^1$, each with vertices at times $t_0, .., t_\tau$, a central trajectory $\mathcal{C}$ has worst case complexity $O(\tau n^2)$.*

Proof. A central trajectory $\mathcal{C}$ is a piecewise function. From Observations 1 and 2 it now follows that $\mathcal{C}$ has a break point at time t only if (a) two subsets of entities become ε-connected or ε-disconnected, or (b) the lower envelope of a set of ε-connected entities has a break point at time t. Within a single time interval $J_i = [t_i, t_{i+1}]$ there are at most $O(n^2)$ times when two entities are at distance exactly ε. Hence, the number of events of type (a) during interval J_i is also $O(n^2)$. By Lemma 7 the total complexity of all lower envelopes of ε-connected sets during J_i is also $O(n^2)$. Hence, the number of break points of type (b) within interval J_i is also $O(n^2)$. The theorem follows. □

4 Computing a Central Trajectory

We now present an algorithm to compute a trajectoid $\mathcal{C}$ minimizing $\mathcal{D}'$. By Lemma 2 such a trajectoid is a central trajectory. The basic idea is to construct a weighted (directed acyclic) graph that represents a set of trajectoids containing an optimal trajectoid. We can then find $\mathcal{C}$ by computing a minimum weight path in this graph.

The graph that we use is a weighted version of the Reeb graph that Buchin et al. [3] use to model the trajectory grouping structure. We review their definition here. The *Reeb graph* $\mathcal{R}$ is a directed acyclic graph. Each edge $e = (u, v)$ of $\mathcal{R}$ corresponds to a maximal subset of entities $C_e \subseteq \mathcal{X}$ that is ε-connected during the time interval $[t_u, t_v]$. The vertices represent times at which the sets of ε-connected entities change, that is, the times at which two entities σ and ψ are at distance ε from each other and the set containing σ merges with or splits from the set containing ψ. See Fig. 3 for an illustration.

By Observation 1 a central trajectory $\mathcal{C}$ can jump from σ to ψ if and only if σ and ψ are ε-connected, that is, if σ and ψ are in the same component C_e of edge e. From Observation 2 it follows that on each edge e, $\mathcal{C}$ uses only the trajectories of entities σ for which f_σ occurs on the lower envelope of the functions $\mathcal{F}_e = \{f_\sigma \mid \sigma \in C_e\}$. Hence, we can then express the cost for $\mathcal{C}$ using edge e by

$$\omega_e = \int_{t_u}^{t_v} \mathcal{L}(\mathcal{F}_e)(t)\,\mathrm{d}t.$$

It now follows that $\mathcal{C}$ follows a path in the Reeb graph $\mathcal{R}$, that is, the set of trajectoids represented by $\mathcal{R}$ contains a trajectoid minimizing $\mathcal{D}'$. So we can compute a central trajectory by finding a minimum weight path in $\mathcal{R}$ from a source to a sink.

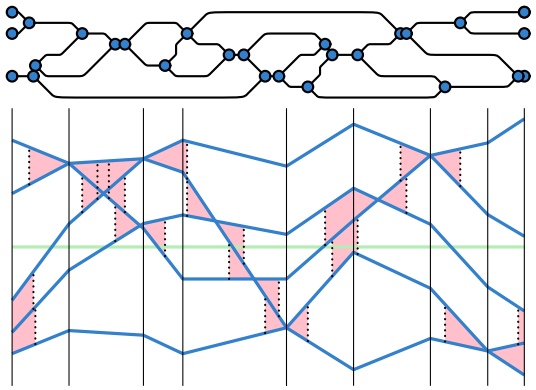


Figure 3: The Reeb graph for a set of moving entities. The dashed lines indicate that two entities are at distance ε.

Analysis. First we compute the Reeb graph as defined by Buchin et al. [3]. This takes $O(\tau n^2 \log n)$ time. Second we compute the weight ω_e for each edge e. The Reeb graph $\mathcal{R}$ is a DAG, so once we have the edge weights, we can use dynamic programming to compute a minimum weight path in $O(|\mathcal{R}|) = O(\tau n^2)$ time. So all that remains is to compute the edge weights ω_e. For this, we need the lower envelope $\mathcal{L}_e$ of each set $\mathcal{F}_e$ on the interval J_e. To compute the lower envelopes, we need the ideal trajectory $\mathcal{I}$, which we can compute $\mathcal{I}$ in $O(\tau n \log n)$ time by computing the lower and upper envelope of the trajectories in each time interval $[t_i, t_{i+1}]$.

Lemma 7 implies that the total complexity of all lower envelopes is $O(\tau n^2)$. To compute them we have two options. We can simply compute the lower envelope from scratch for every edge of $\mathcal{R}$. This takes $O(\tau n^2 \cdot n \log n) = O(\tau n^3 \log n)$ time. Instead, for each time interval $J_i = [t_i, t_{i+1}]$, we compute the arrangement $\mathcal{A}$ representing the modified trajectories on the interval J_i, and use it to trace $\mathcal{L}_e$ in $\mathcal{A}$ for every edge e of $\mathcal{R}$.

Using a standard sweep line algorithm, an arrangement of m line segments can be built in $O((m + A) \log m)$ time, where A is the output complexity. We have $O(n^2)$ line segments: $n + 2$ per entity. Since each pair of trajectories intersects at most once during J_i, we have $A = O(n^2)$. Thus, we build $\mathcal{A}$ in $O(n^2 \log n)$ time. The arrangement represents all break points of type (i), of all functions f_σ. Next, we compute all pairs of points in $\mathcal{A}$ corresponding to break points of type (ii). We do this in $O(n^2)$ time by traversing the zone of $\mathcal{I}$ in $\mathcal{A}$.

We now trace the lower envelopes through $\mathcal{A}$: for each edge $e = (u, v)$ in the Reeb graph with $J_e \subseteq J_i$, we start at the point $\sigma(t_u)$, $\sigma \in C_e$, that is closest to $\mathcal{I}$, and then follow the edges in $\mathcal{A}$ corresponding to $\mathcal{L}_e$, taking care to jump when we encounter break points of type (ii). Our lower envelopes are all disjoint (except at endpoints), so we traverse each edge in $\mathcal{A}$ at most once. The same holds for the jumps. We can avoid costs for searching for the starting point of each lower envelope by tracing the lower envelopes in the right order: when we are done tracing $\mathcal{L}_e$, with $e = (u, v)$, we continue with the lower envelope of an outgoing edge of vertex v. If v is a split vertex where σ and ψ are at distance ε, then the starting point of the lower envelope of the other edge is either $\sigma(t_v)$ or $\psi(t_v)$, depending on which of the two is farthest from $\mathcal{I}$. It follows that when we have $\mathcal{A}$ and the list of break points of type (ii), we can compute all lower envelopes in $O(n^2)$ time. We conclude:

Theorem 9 *Given a set of n trajectories in $\mathbb{R}^1$, each with vertices at times $t_0, .., t_\tau$, we can compute a central trajectory $\mathcal{C}$ in $O(\tau n^2 \log n)$ time using $O(\tau n^2)$ space.*

5 Entities Moving in $\mathbb{R}^d$

For entities moving in $\mathbb{R}^1$ we used that computing a central trajectory was equivalent to finding a trajectoid that minimizes the distance to the ideal trajectory. In $\mathbb{R}^d$, with $d > 1$, however, this is no longer true. Instead, we directly use the functions D_σ expressing the distance between an entity σ and the entity furthest away from σ. We can then still use Observations 1 and 2 to bound the complexity of $\mathcal{C}$ by $O(\tau n^{5/2})$. An algorithm similar to that of Section 4 that runs in $O(\tau n^3)$ time can then be used to compute a central trajectory. The details can be found in the full version [9].

References

[1] P. Bovet and S. Benhamou. Spatial analysis of animals' movements using a correlated random walk model. *J. Theoretical Biology*, 131(4):419–433, 1988.

[2] K. Buchin, M. Buchin, M. van Kreveld, M. Löffler, R. I. Silveira, C. Wenk, and L. Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[3] K. Buchin, M. Buchin, M. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. In *Proc. 2013 WADS Algorithms and Data Structures Symposium*, volume 8037 of *LNCS*, pages 219–230. Springer, 2013.

[4] C. Calenge, S. Dray, and M. Royer-Carenzi. The concept of animals' trajectories from a data analysis perspective. *Ecological Informatics*, 4(1):34 – 41, 2009.

[5] S. Dodge, R. Weibel, and E. Forootan. Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects. *Computers, Environment and Urban Systems*, 33(6):419–434, 2009.

[6] E. Gurarie, R. D. Andrews, and K. L. Laidre. A novel method for identifying behavioural changes in animal movement data. *Ecology Letters*, 12(5):395–408, 2009.

[7] X. Li, X. Li, D. Tang, and X. Xu. Deriving features of traffic flow around an intersection from trajectories of vehicles. In *Proc. 18th Int. Conf. on Geoinf.*, pages 1–5. IEEE, 2010.

[8] A. Stohl. Computation, accuracy and applications of trajectories – a review and bibliography. *Atmospheric Environment*, 32(6):947 – 966, 1998.

[9] M. van Kreveld, M. Löffler, and F. Staals. Central trajectories. *CoRR*, abs/1501.01822, 2015.

Area- and Boundary-Optimal Polygonalization of Planar Point Sets

Sandor Fekete* Stephan Friedrichs† Michael Hemmer* Melanie Papenberg* Arne Schmidt*
Julian Troegel*

Abstract

Given a set of points in the plane, we consider problems of finding polygonalizations that use all these points as vertices and that are minimal or maximal with respect to covered area or length of the boundary. By distinguishing between polygons with and without holes, this results in eight different problems, one of which is the famous Traveling Salesman Problem. Starting from an initial flexible integer programming (IP) formulation, we develop two specific IPs and report preliminary results obtained by our implementation.

1 Introduction

Two of the fundamental structures of Computational Geometry are planar point sets and polygons. Often they come closely related, for example when asking for a *polygonalization*: for a given set V of n points in the plane, find a polygonalization with vertex set V, possibly subject to some objectives and constraints. In the 1990s, the generation of random polygons on a given point set was considered with the motivation of getting polygons as input for geometric algorithms. Auer et al. [1] named five different heuristics for the generation of random polygons. O'Rourke et al. [8] focused on random polygons, while Mitchell et al. [12] worked on random monotone polygons.

Given the geometric character of polygons, natural objectives are *boundary length* and *area*, which may be *minimized* or *maximized*. Furthermore, we may consider *simple polygons* (without holes and self-intersections) or, more generally, *polygons with holes*. Just like that, we have a family of eight basic problems of polygonalizing a set V; see Table 1 and Figure 2.

	general		simple	
	area	boundary	area	boundary
min	MinArea	MinBound	SMinArea	SMinBound
max	MaxArea	MaxBound	SMaxArea	SMaxBound

Table 1: Problem overview

*Department of Computer Science, TU Braunschweig, Germany. s.fekete@tu-bs.de, mhsaar@gmail.com, papenberg.melanie@gmail.com, arne.schmidt@tu-bs.de, j.troegel@tu-bs.de

†Max Planck Institute for Informatics, Saarbrücken, Germany. sfriedri@mpi-inf.mpg.de.

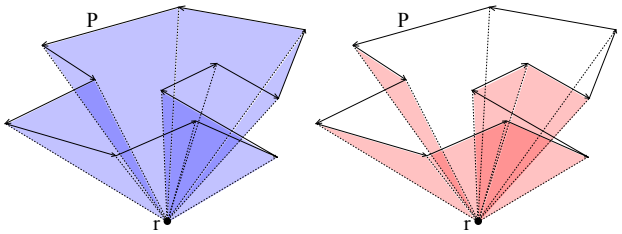


Figure 1: Calculation of the area of P: Using some reference point r, each edge forms an oriented triangle. Counterclockwise triangles contribute positive area (left), others are subtracted (right).

1.1 Related Work on Complexity

Without doubt, the most prominent family member is SMinBound, the Euclidean Traveling Salesman Problem (TSP): Thanks to the triangle inequality, a shortest tour for a given set of vertices is always non-crossing. This classical version of the problem is NP-hard, with well-established benchmark sets [9]. The complexity of MinBound is unknown; note that an optimal 2-factor does not necessarily yield an optimal solution, so the problem may still turn out to be NP-hard. Problems of maximum-boundary length have also been studied, and are related (but not identical) to the Maximum Traveling Salesman Problem [5, 2], whose complexity on a planar point set is Problem #49 of the famous Open Problems Project list [3]. To the best of our knowledge, the complexity of MaxBound as well as SMaxBound is open. Dumitrescu and Tóth [4] gave a $2/\pi$-approximation algorithm for a broad class of instances of SMaxBound. Note that all of these problems have to deal with the additional difficulty of computing a sum of square roots, which is a classical open problem: #33 in [3].

As opposed to the difficulty concerning Euclidean distances, the area of a polygon with rational vertices is rational and can be computed efficiently; see Figure 1. Optimizing area is also a natural problem. While it has received less attention than TSP and its variants, its complexity has been resolved. Fekete [7, 6] showed that SMinArea and SMaxArea are both NP-hard problems; using the same construction, we can conclude that MinArea is also NP-hard. With a separate construction (not given here due to space limitations), we can show that MaxArea is NP-hard as well.

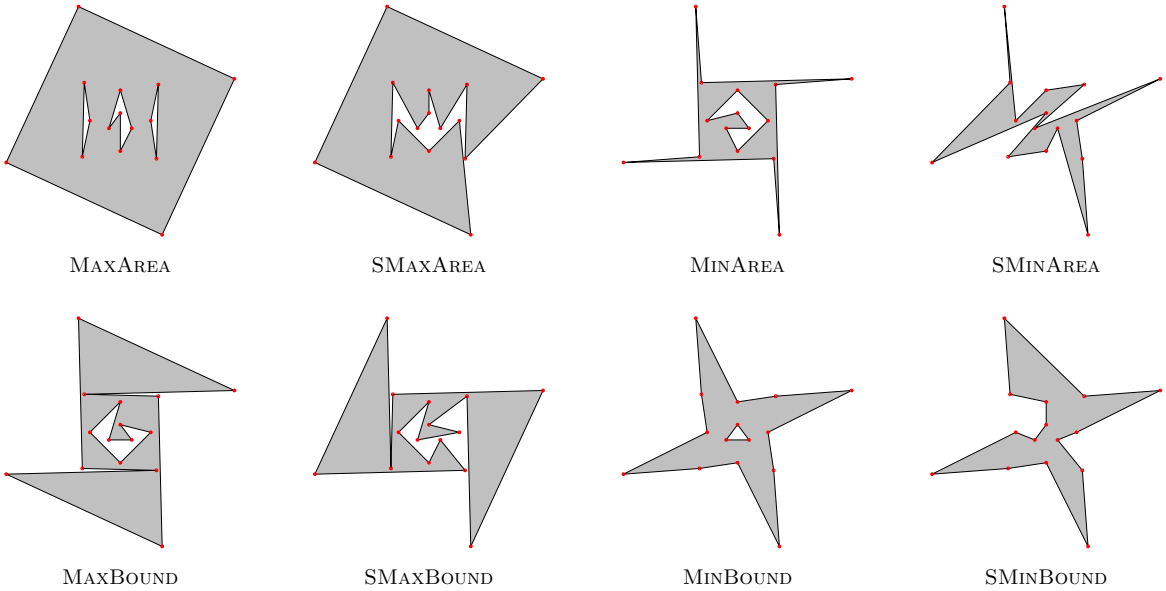


Figure 2: Different optimal solutions on the same input set V for all eight problems.

1.2 Computing Optimal Polygons

As discussed, all problem variants are either known or conjectured to be NP-hard. In the theory community, this is typically used as an incentive for studying approximation algorithms. While several of our problem variants have been studied in this regard (e. g., Euclidean and Max TSP allow polynomial-time approximation schemes, while others allow constant-factor approximations), this is not the goal of this paper. Instead, we want to demonstrate that it is still possible to compute provably optimal solutions for instances of these NP-hard problems, by combining methods of combinatorial optimization (most notably, integer linear programming) with geometric insights. As it turns out, this yields some relatively generic approaches that are suitable for all our problems, and thus possibly for related ones as well. As the ability to check the true optimal values for some instances can provide better comparisons, our approach may also be beneficial for the study of approximation algorithms and heuristics.

2 IP Formulation

Our approach is an IP-based algorithm that solves a relaxation and then, on-demand, adds constraints in a separation phase. (For an introduction to linear and integer programming, see [10].)

2.1 Edge-Based IP for Polygonal Subdivisions

Let $E = L \cup R$ be the set of all oriented edges of the complete graph induced by V, where L and R are the edges directed to the left and right, respectively. By e_{ij} we denote the edge from v_i to v_j and by $z_{ij} \in \{0, 1\}$ its corresponding variable, by $X_R(e_{ij})$ the set of edges in R crossing e_{ij}. Depending on the considered problem, f_{ij} either denotes the Euclidean length of e_{ij} or the signed area of the triangle that is formed by e_{ij} and the origin, which is used as reference point; see Figure 1. Hence, the objective function is given by

$$\min/\max \sum_{i \neq j} f_{ij} z_{ij} + f_{ji} z_{ji} \tag{1}$$

This is subject to the following constraints:

$$\forall i : \sum_{j \neq i} \begin{matrix} z_{ij} = 1 \\ z_{ji} = 1 \end{matrix} \tag{2}$$

$$\forall i \neq j : z_{ij} + z_{ji} \leq 1 \tag{3}$$

$$\forall i < j \text{ and } k_{ij} = |X_R(e_{ij})| : \quad k_{ij} z_{ij} + k_{ij} z_{ji} + \sum_{e_{kl} \in X_R(e_{ij})} (z_{kl} + z_{lk}) \leq k_{ij} \tag{4}$$

$$z_{ij} \in \{0, 1\} \tag{5}$$

Because (2) fixes in- and out-degree of each vertex to one, (3) ensures that for each edge at most one direction is selected, while (4) prevents crossing edges, solving this IP results in an arrangement of non-intersecting oriented (and non-trivial) edge cycles. (1) – (5) generates disjoint, non-trivial, oriented cycles by merely inducing $O(n^2)$ constraints and variables.

However, a polygon is represented by one outer edge cycle, which is oriented counterclockwise and, in the case of general polygons, possibly some inner clockwise cycles that represent the holes. Hence, clockwise cycles incident to the outer face is invalid, as are clockwise cycles that enclose vertices, because they represent holes in holes. Both types are eliminated by the following additional family of constraints.

$$\forall \text{ invalid directed cycles } C : \sum_{\forall e \in C} z_e \leq |C| - 1. \tag{6}$$

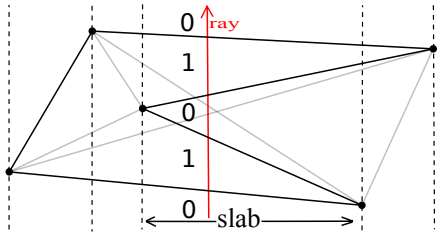


Figure 3: Five points inducing four inner slabs. Winding numbers are indicated along one ray.

However, because there is an exponential number of constraints of type (6), these are added on demand in a separation phase. IP (1) – (6) is named BasicIP.

2.2 A More Efficient Polygonalization

The main issue with BasicIP is that it requires an enormous number of separation steps as it produces many arrangements of boundary cycles that do not represent polygons, let alone a connected or even simple polygon. For example, in the case of MinArea, the IP first prefers boundary cycles with negative area, which then need to be excluded in additional separation steps. It is therefore desirable to enforce a set of boundary cycles that represents a valid polygonal arrangement right away.

Consider a valid polygonal arrangement and a vertical ray from bottom to top. This ray may cross several boundary edges. Let e be a crossed edge. If $e \in R$, the winding number is incremented by one, while it is decremented otherwise; see Figure 3.

Let S be the subdivision of $\mathbb{R}^2$ into $n+1$ vertical slabs induced by V. Shooting a vertical ray along each slab ensures that every face of any arrangement of edge cycles that are induced by V is visited. For slab $s \in S$, we denote by $[e_1^{R_s}, \ldots, e_{K_s}^{R_s}]$ the sequence of edges oriented to the right and crossing s in the order from bottom to top[1], analogously for edges in L. The following pairs of constraints ensure that the winding number alters between 0 and 1 for every slab.

$$\forall s \in S \text{ and } \forall k = 1, \ldots, K_s: \quad \sum_{i=1}^{k} z_{e_i^{R_s}} - z_{e_i^{L_s}} \begin{matrix} \geq 0 \\ \leq 1 \end{matrix} \qquad (7)$$

As k increases to K_s, the sum simulates a walk along the ray from bottom to top, thereby adding or subtracting one to the winding number if a corresponding edge is selected.

This yields an extra $O(n^3)$ constraints as $K_s \in O(n^2)$ and we call the IP (1) – (7) SlabsIP.

[1]It suffices to consider the order of edges with respect to a vertical ray within the slab, because intersecting edges change their order, so they cannot be selected at the same time.

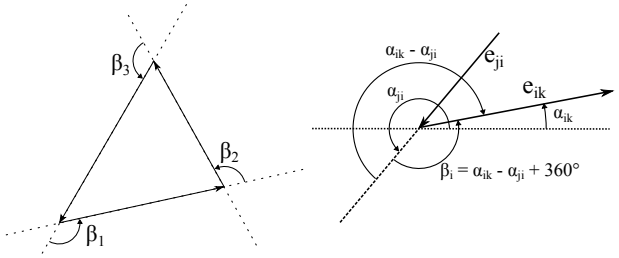


Figure 4: Left: Calculation of Boundary Number. Right: Calculation of enclosing angles.

2.3 Boundary Index

SlabsIP produces valid polygonal arrangements. However, especially when solving SMinArea, many solutions consisting of small distinct polygons have to be eliminated in separation steps. Therefore, it is desirable to add constraints that encode the sum B of positive and negative boundary cycles. In the case of simple polygons, we have $B = 1$, while for general polygons $B \leq 1$ holds.

First observe that for each boundary cycle, we can sum up the angles β_i that are enclosed by two edges at vertex i (see Figure 4). For each counterclockwise cycle, these angles add up to $+360°$, and to $-360°$ for clockwise ones. Hence, summing up at every vertex and dividing by 360° yields B.

Let α_{ij} denote the angle between e_{ij} and the x-axis, in counterclockwise orientation. Now the angle between e_{ij} and e_{jk} at v_j is $\alpha_{jk} - \alpha_{ij}$. This is correct modulo 360°, and we obtain

$$\forall i: \sum_{j \neq i} \alpha_{ij} z_{ij} - \sum_{j \neq i} \alpha_{ji} z_{ji} + 360 y_i \begin{matrix} \leq +180 \\ \geq -180 \end{matrix} \qquad (8)$$

$$y_i \in \{-1, 0, +1\}. \qquad (9)$$

In order to obtain B, we sum up over all angles in (8):

$$B = \frac{1}{360} \sum_{i=1}^{n} \left(\sum_{j \neq i} \alpha_{ij} z_{ij} - \sum_{j \neq i} \alpha_{ji} z_{ji} + 360 y_i \right)$$

However, because we already ensure closed cycles, the α_{ij} cancel out and for general polygonswe only add the following constraint[2]:

$$\sum_{i=1}^{n} y_i \leq 1 \qquad (10)$$

This adds only $O(n)$ constraints and variables. However, note that this does not completely avoid separations steps. For instance, an arrangement of two polygons, one of which has one hole, has boundary index one. Such a solution must still be cut off in a separation step. IP (1) – (10) is named BIndexIP.

[2]For simple polygons, we use $\sum_{i=1}^{n} y_i = 1$

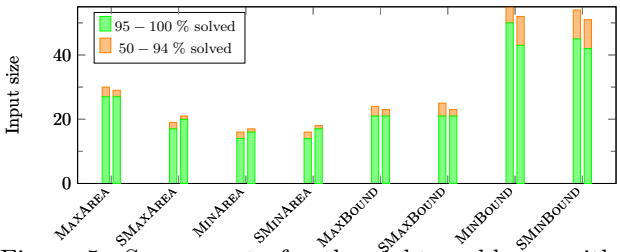


Figure 5: Success rate for the eight problems, with 30 instances for each input size (y-axis). Left bars: SlabsIP. Right bars: BIndexIP.

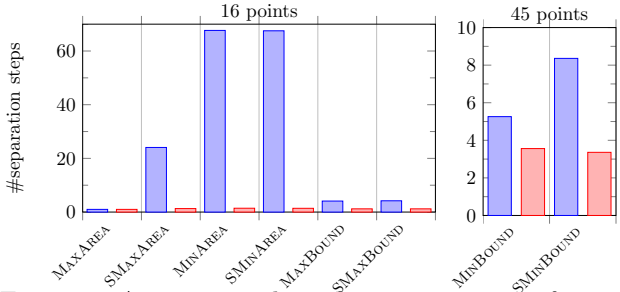


Figure 6: Average number of separation steps for 30 instances of the given input size. All instances where solved within the time limit. Blue: SlabsIP. Red: BIndexIP.

3 Experiments

Our implementation uses CPLEX to represent and solve the presented IPs. The geometric part is based on the CGAL Arrangements package [11]. CGAL represents planar subdivision by a doubly connected edge list (DCEL), which is ideal for detecting invalid boundary cycles.

All experiments were run on an *Intel Core i7-4770* CPU clocked at 3.40 GHz with 16 GB of RAM. For each point size we considered 30 randomly generated instances, with a time limit of 30 minutes. We do not show benchmark results for BasicIP, as it performed much worse than SlabsIP and BIndexIP.

We observe that both generic IPs perform much better for the minimum boundary problems (MinBound and SMinBound, i.e., the TSP), for which we are able to solve all instances up to about 45 input points; see Figure 5. This is still much worse than the performance of custom-made approaches for the TSP, but it illustrates the greater practical difficulty of the other problems. For all problems that optimized boundary BIndexIP, performed worse than SlabsIP, as BIndexIP was not able to significantly reduce the number of separation steps. In the case of MaxBound and SMaxBound, already SlabsIP requires hardly any separation steps; see Figure 6.

In the case of problems optimizing the area, we can observe that BIndexIP indeed improves the performance significantly. The only exception is MaxArea, for which already SlabsIP usually does not require any separation step. The reason is that the optimal result is essentially the convex hull of the point set with some small inner triangles removed, which is usually found in the first round. However, for SMaxArea as well as MinArea and SMinArea we can observe a significant advantage for BIndexIP, as it is essentially able to skip the separation step.

Why are minimum boundary problems practically easier to solve? For these problem variants, the intersection constraints (4) are already implied by triangle inequality, and as such only give a slight overhead; this is not the case for the other problem variants. This can be further exploited; for problems optimizing the boundary, a more specialized IP formulation can be based on undirected edges, requiring only half the number of variables as in our generic approach.

References

[1] T. Auer and M. Held. Rpg: Heuristics for the generation of random polygons. In *8th Canadian Conference on Computational Geometry*, pages 38–44, 1996.

[2] A. I. Barvinok, S. P. Fekete, D. S. Johnson, A. Tamir, G. J. Woeginger, and R. Wodroofe. The geometric maximum Traveling Salesman Problem. *J. ACM*, 50:641–664, 2003.

[3] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problems project, 2001. `http://cs.smith.edu/~orourke/TOPP/`.

[4] A. Dumitrescu and C. D. Tóth. Long non-crossing configurations in the plane. pages 727–752, 2010.

[5] S. P. Fekete. Simplicity and hardness of the maximum Traveling Salesman Problem under geometric distances. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 337–345, 1999.

[6] S. P. Fekete. On simple polygonalizations with optimal area. *DCG*, 23(1):73–110, 2000.

[7] S. P. Fekete and W. R. Pulleyblank. Area optimization of simple polygons. In *Proc. 9th Annu. ACM Sympos. Sympos. Geom.*, pages 173–182, 1993.

[8] J. O'Rourke and M. Virmani. Generating random polygons. Technical Report 11, Dept. Comput. Sci., Smith College, 1991.

[9] G. Reinelt. TSPlib – A Traveling Salesman Problem library. *ORSA J. on Computing*, 3(4):376–384, 1991.

[10] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[11] R. Wein, E. Berberich, E. Fogel, D. Halperin, M. Hemmer, O. Salzman, and B. Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.3 edition, 2014.

[12] C. Zhu, G. Sundaram, J. Snoeyink, and J. S. Mitchell. Generating random polygons with given vertices. *Computational Geometry*, 6(5):277 – 290, 1996. Sixth Canadian Conference on Computational Geometry.

Pruning Oracles for High-Dimensional Vector Sets

Stefan Funke [i] Sabine Storandt [ii]

Abstract

We consider the following problem: Given a set S of non-negative d-dimensional cost vectors, we want to compute $S' \subseteq S$ with S' containing only vectors v from S for which exists an $\alpha \in \mathbb{R}^d_{\geq 0}$ such that $\alpha^T v \leq \alpha^T w$ for all $w \in S$. Based on a geometric interpretation of this problem we propose pruning oracles which reduce S to S'. We outline how these oracles can be employed in practice in the context of personalized route planning in huge street networks. Finally, we prove the effectiveness of the oracles experimentally using a CGAL-based implementation.

1 Introduction

Given a set S of non-negative d-dimensional vectors, we want to solve the minimization problem $\min_{v \in S} \alpha^T v$ with $\alpha \in \mathbb{R}^d_{\geq 0}$. In particular, we want to answer the following type of query: for given α, find the vector $v \in S$ which yields the minimum value $\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_d v_d$. Obviously, if S contains vectors that are *never* optimal no matter how α is chosen, these vectors can be pruned from S. We call the reduced set S'. The more concise S', the faster the optimal vector can be identified for given α, and the less space is needed to store S'. Hence our goal is to construct pruning oracles, which reduce S to the smallest possible subset S', such that for any choice of α the solution is still the same as for S.

If we interpret the d-dimensional vectors in S as points in a d-dimensional coordinate system, the vectors that have to be kept in S' are exactly the ones that correspond to corners of the convex hull of S that are visible from the origin of the coordinate system (for an opaque convex hull). In the dual view, where the vectors become hyperplanes, these are the ones on the lower envelope of all hyperplanes. In Figure 1 both views are illustrated for $d = 2$.

In principle, the interpretation of vectors as points in $\mathbb{R}^d$ provides us with a straightforward solution for our problem. In fact, we only have to construct the convex hull and inspect the part of it that is visible from the origin. But in practice this approach is not really applicable. The complexity of the boundary description of the convex hull of $|S|$ points in d dimensions might be in the order of $|S|^{\lfloor d/2 \rfloor}$. Already for moderate dimensions, the computation of the convex hull becomes very expensive for large sets S and also error-prone if floating point arithmetic is used. Experimenting with an implementation of a d-dimensional convex hull algorithm in CGAL, we could not get acceptable running times for d larger than 3.

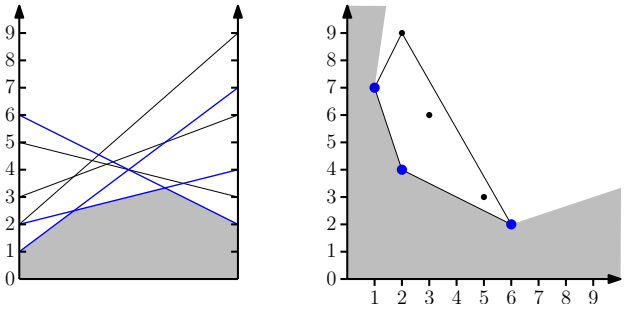


Figure 1: Two representations of the vector set $(1,7),(2,4),(2,9),(3,6),(5,3),(6,2)$; in the left image as lines, in the right image as points. The items that border the visible area of interest (gray) are coloured blue. Those form the boundary of the convex hull visible from the origin. In this example it is $(1,7),(2,4),(6,2)$.

The scope of this paper is the design of efficient methods to construct S' (or small supersets thereof) without explicitly constructing the convex hull of S.

1.1 Related Work

There exist various methods to construct the convex hull in d dimensions, e.g. a deterministic algorithm with a runtime of $\mathcal{O}(|S| \log |S| + |S|^{\lfloor d/2 \rfloor})$ [2], or the QuickHull algorithm [1]. But these methods are designed for computing and investigating the complete facet structure of the convex hull. In our scenario, we are neither interested in the part of the convex hull not visible from the origin nor in the actual facet structure, but only in the visible extreme points.

1.2 Contribution

We present several pruning strategies which reduce the set S of d-dimensional cost vectors efficiently. We show that these pruning oracles can be employed in the context of personalized route planning in huge street networks. Our experimental results prove that our oracles work in practice and can be implemented

[i]Department of Computer Science, Universität Stuttgart, Germany, funke@fmi.uni-stuttgart.de
[ii]Department of Computer Science, Universität Freiburg, Germany, storandt@informatik.uni-freiburg.de

(using CGAL) even for high dimensions d.

2 Pruning High-Dimensional Cost Vectors

Let us first focus on pruning strategies/oracles which might not lead to the minimal possible reduced set S', but are simple and very efficient. Then we introduce a more sophisticated incremental approach, which constructs the optimal S'.

2.1 Pruning Dominated Vectors

The most straightforward strategy to prune vectors from S is by domination. A vector $v \in \mathbb{R}^d$ dominates another vector $w \in \mathbb{R}^d$ if $v_i \leq w_i \ \forall i = 1, \ldots, d$ and $v_i < w_i$ for at least one $i \in \{1, \ldots, d\}$.

One can compare each vector v to every other vector v' in the set, checking component-wise if $v_i \leq v'_i$ for $i = 1, \cdots, d$ in order to prune out all dominated vectors. This leads naively to a runtime of $\mathcal{O}(d|S|^2)$.

2.2 Pruning Spanned Vectors

Of course, not every non-dominated vector is optimal for some choice of α. The following Lemma characterizes superfluous, non-dominated cost vectors that can never be optimal no matter what α has been chosen.

Lemma 1 *A vector $v \in \mathbb{R}^d$ can be pruned from a set of vectors S if a convex combination v' of at most d other vectors from S dominates v.*

Proof. Assume for contradiction that for given α, vector v uniquely defines the minimum cost $\alpha^T v = z$, that is, v cannot be pruned. Let $w_1, w_2, \cdots, w_d$ be the d vectors that span v' which dominates v. So we can represent v' as $\gamma_1 w_1 + \gamma_2 w_2 + \cdots + \gamma_d w_d$ with $\sum_{i=1}^{d} \gamma_i = 1$, $\gamma_i \geq 0$. By the domination property we know that $\alpha^T(\gamma_1 w_1 + \gamma_2 w_2 + \cdots + \gamma_d w_d) \leq z$ as well. This formula can be rearranged as follows:

$$\gamma_1 \alpha^T w_1 + \gamma_2 \alpha^T w_2 + \cdots + \gamma_d \alpha^T w_d \leq z$$

As $\alpha^T v = z$ is the minimum among all possible vectors, we conclude that $\alpha^T w_i > z$ for $i = 1, \cdots, d$. Plugging this observation in the formula above, we get:

$$\sum_{i=1}^{d} \gamma_i \alpha^T w_i > \sum_{i=1}^{d} \gamma_i z = z \sum_{i=1}^{d} \gamma_i = z$$

This obviously contradicts the fact that the left hand side is less or equal to z. Hence the initial assumption that v is necessary to get the minimum cost for some α is wrong, and v can be pruned. □

The other direction is also easy to see. If a vector can $v \in \mathbb{R}^d$ can be pruned from S, i.e., for any α there is a better vector in S, the respective vectors convexly combine to a vector dominating v.

2.2.1 A Simple Pruning Oracle

Lemma 1 tells us that a vector v can be pruned if it is dominated by a vector v' spanned by d other vectors. Given a set of d vectors $w_1, w_2, \cdots, w_d$, the test whether a vector v is pruned by them boils down to checking whether the following system of linear inequalities has at least one feasible solution:

$$\begin{aligned}
\gamma_1 w_{11} + \gamma_2 w_{21} + \cdots + \gamma_d w_{d1} &\leq v_1 \\
\gamma_1 w_{12} + \gamma_2 w_{22} + \cdots + \gamma_d w_{d2} &\leq v_2 \\
&\cdots \\
\gamma_1 w_{1d} + \gamma_2 w_{2d} + \cdots + \gamma_d w_{dd} &\leq v_d \\
\gamma_1 + \gamma_2 + \cdots + \gamma_d &= 1 \\
\gamma_i &\geq 0
\end{aligned}$$

A straightforward pruning oracle then checks for each vector if it is pruned by one of the $b = \binom{|S|}{d}$ possible choices of d vectors from S. This issues $O(b \cdot |S|)$ calls to a linear programming solver, rendering this approach somewhat impractical. We can't do wrong by only pruning with some few promising bases, though.

A natural choice for a promising set of d vectors is to pick for each dimension the vector with minimum value in the respective component. If by that a vector is chosen for several dimensions, vectors with second, third, ... smallest value in the respective components are chosen from S. In practice, the resulting set of vectors quickly prunes a large fraction of superfluous vectors out of S.

2.2.2 A Complete Pruning Oracle

As mentioned before, the vectors/points that have to be kept in S' are exactly the ones visible from the origin on the boundary of the convex hull of all vectors/points. As we are only interested in those points, we have no use for the complete facet structure of the convex hull boundary. So the question is, how to determine the desired points efficiently without wasting too much time for constructing parts of the convex hull that we never inspect anyway.

To guarantee that the complexity of the final convex hull description is only influenced by the part visible from the origin, we simply augment S with d auxiliary points $p_1, \cdots, p_d \in \mathbb{R}^d$ where every component of p_i equals 0 except for $p_{ii} = \infty$. Obviously, all non-auxiliary extreme points of the convex hull of the augmented point set are visible extreme points of the convex hull of the original point set.

When applying one of the standard convex hull algorithms, the intermediate description of the convex hull during the construction might be significantly more complex than the final structure. To avoid such an overhead as far as possible, and at the same time accelerate the computation of the visible points, we devise an incremental approach. Starting with the

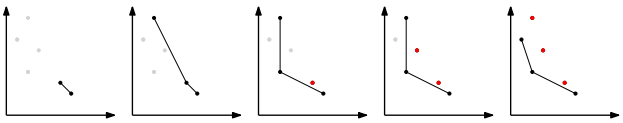


Figure 2: Illustration of our incremental pruning oracle for $d = 2$. Gray points are untouched so far. Points already considered are black if they lie on the boundary of the convex hull and are visible from the origin, or red if already pruned.

auxiliary vectors we add the points from S one by one, points more promising to be extreme first. Whenever a point is added, we have to decide whether it is extreme with respect to the points considered so far. With the current extreme points, we can instrument a linear program similar to the one introduced in the last subsection to check for extremeness. If the new point/vector is spanned, it can be discarded. Otherwise the previously inserted points have to be checked for extremeness again. This approach issues $O(|S|^2)$ calls to the extremeness oracle. It is reasonably efficient in practice as long as the number of visible points is not too big. Clearly, any vector v that is spanned by some other vectors is pruned as soon as all these vectors and v have been considered and vice versa, only superfluous vectors can be pruned.

Figure 2 illustrates this incremental pruning oracle on the example instance used in Figure 1.

3 Application: Personalized Route Planning

When planning the best route from A to B in a street network, the notion of optimality differs from person to person. Some only care about reaching their destination as quickly as possible, others want to minimize fuel consumption, avoid tolls, or prefer scenic routes. Often a fair trade-off between several such preferences is the desired route. We call this problem *personalized route planning (PRP)* and formalize it as follows:

For a street network $G(V, E)$, we have for each edge $e \in E$ a d-dimensional cost vector $c(e) \in \mathbb{R}^d_{\geq 0}$ (e.g. c_1 corresponding to travel time, c_2 to gas price, etc.). A query consists not only of source and target $s, t \in V$ but also of non-negative weights $\alpha_1, \alpha_2, \cdots, \alpha_d$, where α_i expresses the importance of edge cost component i for the user. The goal is to compute the path p from s to t in G which minimizes $\sum_{e \in p} \alpha^T c(e)$.

Dijkstra can compute personalized routes for given α, by evaluating $\alpha^T c(e)$ on demand when relaxing an edge. A Dijkstra run takes in the order of several seconds on continental sized road networks, though. In [3], an acceleration scheme based on k-Path Covers was proposed which achieves a speed-up of about 13.

A k-Path Cover (k-PC) on a graph $G(V, E)$ is a subset of the nodes $W \subseteq V$, such that for every simple path in G consisting of k nodes at least one of those nodes is contained in W, see Figure 3 for an example.

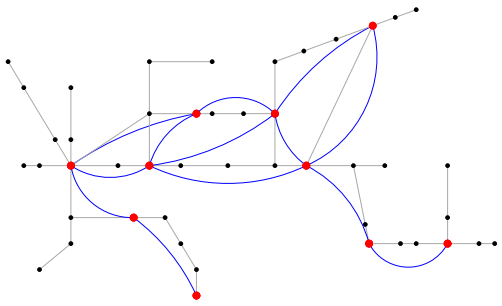


Figure 3: Example of a k-Path Cover for $k = 4$ (red nodes), and the induced overlay graph (blue edges).

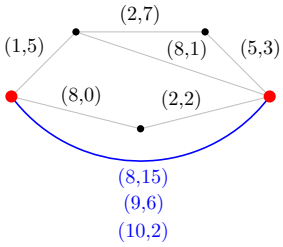


Figure 4: Example for two metrics ($d = 2$). Cover nodes are colored red, the overlay graph consisting of a single edge is blue. As there are three different simple paths from one cover node to the other, the overlay edge between them gets assigned three labels. The overlay labels stem from summing up the original labels along the respective paths.

k-Path Covers can be instrumented for personalized route planning as follows. First, an overlay graph is computed which contains an edge between any two neighbors in the cover. Here, two cover nodes are neighbors if there exists a simple path between them in G not containing any other cover node (see again Figure 3 for an illustration). Then every edge (u, v) in the overlay graph is augmented with cost vectors. In particular, for every simple path p from u to v free of other cover nodes, the accumulated cost vector $\sum_{e \in p} c(e)$ is associated with (u, v). A small example of this process in provided in Figure 4. This completes the preprocessing phase. Note that the number of vectors associated with an edge (and hence the space consumption) grows quickly with k as the number of paths between two nodes can grow exponentially with the number of nodes in the network.

In a query, first local Dijkstra computations are run from s and t (reversely) until all paths in the Dijkstra search tree contain at least one settled cover node. The remaining search between nodes settled in the runs from s and t is conducted in the overlay graph only. During the search edge costs are evaluated using α provided with the query. The query times are dominated by the search in the overlay graph, in particular by the number of edges and cost vectors assigned to

			unpruned	dominance		simple		complete		simple + complete	
	k	$\|C\|$	$\sum \|S\|$	$\sum \|S\|$	time(s)	$\sum \|S\|$	time(s)	$\sum \|S\|$	time(s)	$\sum \|S\|$	time(s)
BW	12	256k	1,385k	1,084k	<1	1,084k	<1	1,083k	1,557	1,083k	616
	20	161k	1,755k	941k	<1	941k	<1	935k	2,535	935k	911
	24	137k	2,171k	916k	1	916k	1	908k	2,904	908k	1,040
	32	107k	4,484k	909k	18	911k	17	894k	4,745	894k	1,199
	40	90k	20,883k	923k	6,215	925k	26	-	-	899k	1,465
	48	78k	98,690k	-	-	959k	111	-	-	919k	1,666
GER	20	1,064k	12,000k	6,176k	4	6,176k	3	-	-	6,142k	5,942
	24	876k	18,494k	5,970k	38	5,970k	13	-	-	-	-
	32	710k	40,709k	5,911k	1,051	5,917k	65	-	-	-	-

Table 1: Pruning of vector sets in the k-PC based overlay graph, $|C|$ denotes the size of the k-Path Cover, $\sum |S|$ denotes the total number of cost vectors generated in the overlay graph. Timings are given in seconds.

these edges. So while in principle large values of k are desirable to reduce the number of nodes in the overlay graph and the query times, the extreme blow-up in the number of cost vectors limited the usability of this approach to values of k around 20. Now the obvious idea is to prune the set of cost vectors using the oracles described in the last section. Decreasing the number of cost vectors significantly should lead to more efficient personalized route planning queries allowing both more metrics and/or larger values of k.

4 Experimental Results and Future Work

Our implementation of the k-Path Cover based personalized route planning approach is written in C++. For the pruning oracles we used the Computational Geometry Algorithms Library (CGAL) [4], in particular their exact linear programming solver. Timings were measured on a single core of an Intel(R) i7-3930K CPU with 3.20GHz and 64GB RAM.

We present results on two large graphs: Baden-Württemberg (BW) with 2.23 million edges and and 4.64 million edges, and Germany (GER) with 17.73 million nodes and 36.06 million edges (extracted from OSM[1]). We applied the k-Path Cover approach [3] to both graphs for varying values of k and constructed the overlay graph and the respective initial vector sets S for each edge using $d = 8$ natural metrics. Then we applied our proposed pruning strategies and measured the resulting set sizes. In Table 1, the values are summarized. We observe that simple pruning can be applied to all our instances, while the complete pruning oracle becomes impractical for BW and $k = 40$. If we apply simple pruning first, and then the complete pruning oracle on top, we can solve all BW instances and even GER for $k = 20$. In general, our pruning oracles reduce the set sizes for all instances significantly. The effect becomes more pronounced with increasing value of k, as the number of simple paths between two nodes grows quickly with k, so the potential number of non-optimal cost vectors in S is higher. For BW and $k = 48$ less than 1% of the initial cost vectors remain in the final sets. In terms of space, the overlay graph reduces from 24.6GB to 161MB after pruning.

Runtimes reported for dominance pruning correspond to the naive quadratic implementation. So there is possibly a lot of room for acceleration here, especially as we could use dominance pruning also in an incremental fashion. We conducted experiments for even larger k than presented in Table 1 on the bases of a subsample of overlay graph edges and not the whole graph, though. These experiments indicate that the efficacy (in terms of result size) of dominance pruning compared to complete pruning deteriorates rapidly. Complete pruning outperformed dominance pruning by a factor of 2 to 28.

Hence in future work, dominance pruning should be made more efficient, but mainly to allow for an initial reduction of the set that is then fed into another (complete) pruning oracle. Here, either our proposed incremental complete oracle should be accelerated (e.g. by coming up with a clever order of the points to add), or new (complete) oracles should be designed that allow to tackle even larger vector sets efficiently. Also, we currently employ an *exact* LP solver as not to compromise optimality of the resulting paths. It might be worth to investigate a conservative use of fast floating-point LP solvers as long as they do not destroy optimality of the PRP queries.

[1] `https://www.openstreetmap.org/`

References

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[2] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(1):377–409, 1993.

[3] S. Funke, A. Nusser, and S. Storandt. On k-path covers and their applications. *Proceedings of the VLDB Endowment*, 7(10), 2014.

[4] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.4 edition, 2014.

Non-crossing Monotonic Paths in Labeled Point Sets on the Plane

Toshinori Sakai*† Jorge Urrutia‡§

Abstract

Let n be a positive integer, and let P be a set of n points in general position on the plane with labels $1, 2, \ldots, n$. The label of each $p \in P$ will be denoted by $\ell(p)$. A polygonal line connecting k elements $p_1, p_2, \ldots, p_k$ of P in this order is called a *monotonic path of length* k if the sequence $\ell(p_1), \ell(p_2), \ldots, \ell(p_k)$ is monotonically increasing or decreasing in this order. We show that P contains a vertex set of a *non-crossing* monotonic path of length at least $c(\sqrt{n}-1)$, where $c = 1.0045\ldots$.

1 Introduction

Let P be a set of points on the plane. P is *in general position* if no three of its elements are collinear. Furthermore, P is *in convex position* if all points are vertices of the convex hull of P. All point sets P considered in this paper are in general position, and consisting of points with pairwise different labels $1, 2, \ldots, |P|$. We will refer to these point sets as *lp*-sets. For each *lp*-set P, the label of a point $p \in P$ will be denoted by $\ell(p)$.

Let P be an *lp*-set. A polygonal line connecting k elements $p_1, \ldots, p_k$ of P in this order is called a *monotonic path of length* k if the sequence $\ell(p_1), \ldots, \ell(p_k)$ is monotonically increasing or decreasing (Figure 1). When P contains the vertex set of a non-crossing monotonic path of length k, we will say that P *contains* a non-crossing monotonic path of length k.

The *length* of a finite sequence is the number of its terms. The following theorem is (a corollary of) a well known result by Erdős and Szekeres [3]:

Theorem 1 *Let n be a positive integer. Then any sequence of n distinct real numbers contains a monotonically increasing or decreasing subsequence of length at least $\sqrt{n}$. This bound is tight.*

In [4], Sakai and Urrutia proved that any n-element *lp*-set in *convex* position contains a non-crossing monotonic path of length at least $\sqrt{3n-\frac{3}{4}}-\frac{1}{2}$, improving on a result by Czyzowicz, Kranakis, Krizanc and Urrutia [2]. In [4], it is also conjectured that any n-element *lp*-set in convex position contains a non-crossing monotonic path of length at least $2\sqrt{n}-1$.

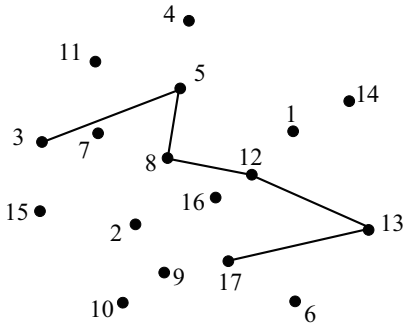


Figure 1: An *lp*-set (each number represents the label of each element) and a monotonic path of length 6.

Furthermore, it has been believed that there exists a constant $c > 1$ such that the following statement holds: any n-element *lp*-set in *general* position contains a non-crossing monotonic path of length at least $c\sqrt{n} - o(\sqrt{n})$. In Section 2, we show the following result:

Theorem 2 *Let n be a positive integer. Then any n-element lp-set P in general position contains a non-crossing monotonic path of length at least $c(\sqrt{n}-1)$, where* $c = \frac{1}{2}\left(\sqrt{\sqrt{\frac{10}{3}}-1} + \frac{1}{\sqrt{\sqrt{\frac{10}{3}}-1}}\right) = 1.0045\ldots$.

Note that it is easy to verify that any n-element *lp*-set contains a non-crossing monotonic path of length at least $\sqrt{n}$. Actually, we have only to take a straight line l that is not perpendicular to any straight line connecting two distinct elements of P, to project all elements of P orthogonally to l, and to apply Theorem 1 to the sequence obtained on l. Though the constant $c = 1.0045\ldots$ in Theorem 2 is just slightly greater than 1, the result shows that the behavior of problems on monotonic sequences and non-crossing monotonic paths are essentially different.

2 Proof of Theorem 2

In this section, we prove Theorem 2 (for $n \geq 4$).

*Department of Mathematics, School of Science, Tokai University, Japan. sakai@tokai-u.jp

†Research supported by JSPS KAKENHI Grant Number 24540144.

‡Instituto de Matemáticas, Ciudad Universitaria, Universidad Nacional Autónoma de México, México D.F., México. urrutia@matem.unam.mx

§Research partially supported by CONACyT(Mexico) grant CB-2012-01-0178379.

A finite sequence $\{x_i\}_{i=1}^n$ is said to be *unimodal* (resp. *anti-unimodal*) if there is an m, $1 \le m \le n$, such that $x_1 < x_2 < \cdots < x_m$ and $x_m > x_{m+1} > \cdots > x_n$ (resp. $x_1 > x_2 > \cdots > x_m$ and $x_m < x_{m+1} < \cdots < x_n$). To prove Theorem 2, we use the following Theorem 3 which was first obtained by Chung [1], and later by Sakai and Urrutia [4].

Theorem 3 *Let n be a positive integer. Then any sequence of n distinct real numbers contains a unimodal or anti-unimodal subsequence of length at least $\sqrt{3n - \frac{3}{4}} - \frac{1}{2}$.*

In Figure 2, for $k = 3$, we present an example with $n = 3k^2 + k = 30$ terms whose longest unimodal/anti-unimodal subsequence has length $\left\lceil \sqrt{3n - \frac{3}{4}} - \frac{1}{2} \right\rceil = 3k = 9$.

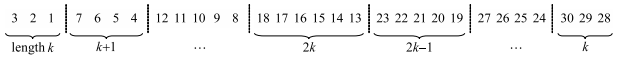


Figure 2: The maximum length of a unimodal/anti-unimodal subsequence is $3k$.

Now we proceed to the proof of Theorem 2. We may assume that P is an *lp*-set on $\mathbb{R}^2$, and that no two points of P have the same x-coordinate. Let $p_1, p_2, \ldots, p_n$ be the elements of P in increasing order of their x-coordinates, and let $\mathcal{L}$ be the sequence $\ell(p_1), \ell(p_2), \ldots, \ell(p_n)$ (recall that $\ell(x)$ denotes the label of point x), which is a permutation of $\{1, 2, \ldots, n\}$. For each i with $1 \le i \le n$, let a_i denote the length of the longest *increasing* subsequences of $\mathcal{L}$ *ending* at $\ell(p_i)$, b_i the length of the longest *decreasing* subsequences of $\mathcal{L}$ ending at $\ell(p_i)$, and A_i the point (a_i, b_i) on the ab-coordinate plane. Set $\mathcal{A} = \{A_i : 1 \le i \le n\}$. We can verify that the following lemma holds:

Lemma 4 *Let i and j be integers with $1 \le i < j \le n$. Then the following (i) and (ii) hold.*
(i) If $\ell(p_i) < \ell(p_j)$, then $a_j \ge a_i + 1$.
(ii) If $\ell(p_i) > \ell(p_j)$, then $b_j \ge b_i + 1$.

So, for distinct indices i and j, we must have $A_i \ne A_j$.

First consider the case where there exists m such that $a_m \ge c(\sqrt{n} - 1)$ (recall that $c = 1.0045\ldots$, as in the statement of Theorem 2). In this case, there exists a non-crossing path connecting a_m points of P and ending at p_m such that the values of the labels of its vertices monotonically increase along it, as desired. Also, in the case where there exists m such that $b_m \ge c(\sqrt{n} - 1)$, we can find a path with desired properties as well. Thus we may assume that

$$\left.\begin{array}{l} a_i < c(\sqrt{n} - 1) \text{ and } b_i < c(\sqrt{n} - 1) \\ \text{for all } 1 \le i \le n. \end{array}\right\} \quad (1)$$

A Non-crossing Monotonic Path $\mathcal{P}$

Let $d = \sqrt{\sqrt{\frac{10}{3}} - 1} = 0.9087\ldots$. We have $c = \frac{1}{2}\left(d + \frac{1}{d}\right)$, and hence

$$2cd = d^2 + 1. \quad (2)$$

We can also verify the following (3) and (4).

$$0.09 < c - d < 0.1. \quad (3)$$
$$14c^2 - 5d^2 = 10. \quad (4)$$

Lemma 5 *There exists m such that*

$$a_m > d(\sqrt{n} - 1) \text{ and } b_m > d(\sqrt{n} - 1) \quad (5)$$

(Figure 3).

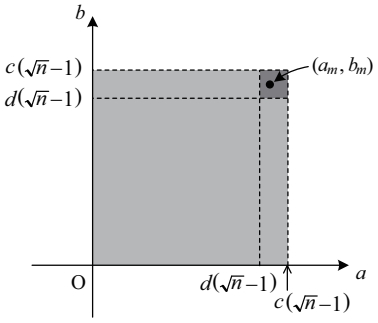


Figure 3

Proof. By way of contradiction, suppose that $a_i \le d(\sqrt{n} - 1)$ or $b_i \le d(\sqrt{n} - 1)$ for all i. From this assumption and (1), it follows that

$$\begin{aligned} |\mathcal{A}| &< [c(\sqrt{n} - 1)]^2 - [(c - d)(\sqrt{n} - 1) - 1]^2 \\ &< n - 2\sqrt{n} + 2(c - d)(\sqrt{n} - 1) \quad \text{(by (2))} \\ &< n \quad \text{(by (3))}, \end{aligned}$$

a contradiction. □

Take m satisfying (5). By symmetry, we may assume that

$$\ell(p_m) \le \frac{n}{2}. \quad (6)$$

Also, by the definition of the a_i, there is a non-crossing path $\mathcal{P}$ connecting a_m points of P and ending at p_m such that the values of the labels of points monotonically increase along $\mathcal{P}$. We have

$$\text{the length of } \mathcal{P} = a_m > d(\sqrt{n} - 1) \quad (7)$$

by (5).

A Path Connecting a Unimodal Sequence

Next we define Q_1 and Q_2 by

$$Q_1 = \{p_i : 1 \le i \le m-1 \text{ and } \ell(p_i) > \ell(p_m)\}, \text{ and}$$
$$Q_2 = \{p_i : m+1 \le i \le n \text{ and } \ell(p_i) > \ell(p_m)\}$$

(so, in particular, the x-coordinates of the elements of Q_1 (resp. Q_2) are smaller (resp. greater) than the x-coordinate of p_m). By Lemma 4 (i) and (5), $a_i \ge a_m + 1 > d(\sqrt{n}-1)+1$ for any $p_i \in Q_2$. From this and (1), it follows that for any $p_i \in Q_2$,

$$\begin{aligned} d(\sqrt{n}-1)+1 &< a_i < c(\sqrt{n}-1) \quad \text{and} \\ 1 &\le b_i < c(\sqrt{n}-1), \end{aligned}$$

and hence

$$\begin{aligned} |Q_2| &< (c-d)(\sqrt{n}-1) \times c(\sqrt{n}-1) \\ &= c(c-d)(\sqrt{n}-1)^2. \end{aligned}$$

From this, we obtain

$$\begin{aligned} |Q_1| &= (n-\ell(p_m)) - |Q_2| \\ &> \frac{n}{2} - c(c-d)(\sqrt{n}-1)^2 \\ &> \frac{1}{7}\left(\sqrt{\frac{10}{3}}+1\right)n + \frac{1}{4} \\ &= \frac{1}{3d^2}n + \frac{1}{4} \end{aligned} \tag{8}$$

by (2), (3), (4) and the assumption that $n \ge 4$.

Connect p_m and each element of Q_1, and relabel the elements of Q_1 as $q_1, q_2, \ldots, q_{|Q_1|}$ in the counterclockwise order around p_m. We choose q_1 in such a way that all other elements of Q_1 lie on the left side of directed line $p_m q_1$.

By Theorem 3 and (8), there exists a path $\mathcal{Q} = q_{i_1} q_{i_2} \ldots q_{i_k}$ of length

$$k \ge \sqrt{3|Q_1| - \frac{3}{4}} - \frac{1}{2} > \frac{1}{d}\sqrt{n} - \frac{1}{2} \tag{9}$$

such that $i_1 < i_2 < \cdots < i_k$, and such that either

(i) $\ell(q_{i_1}) < \cdots < \ell(q_{i_h}) > \ell(q_{i_{h+1}}) > \cdots > \ell(q_{i_k})$ or
(ii) $\ell(q_{i_1}) > \cdots > \ell(q_{i_h}) < \ell(q_{i_{h+1}}) < \cdots < \ell(q_{i_k})$

holds for some h. Define *monotonic* subpaths $\mathcal{R}_1$ and $\mathcal{R}_2$ by

$$\begin{aligned} \mathcal{R}_1 &= q_{i_1} q_{i_2} \ldots q_{i_h} \text{ and} \\ \mathcal{R}_2 &= q_{i_h} q_{i_{h+1}} \ldots q_{i_k}, \end{aligned}$$

and also define ${\mathcal{R}_1}^{-1}$ and ${\mathcal{R}_2}^{-1}$ by

$$\begin{aligned} {\mathcal{R}_1}^{-1} &= q_{i_h} q_{i_{h-1}} \ldots q_{i_1} \text{ and} \\ {\mathcal{R}_2}^{-1} &= q_{i_k} q_{i_{k-1}} \ldots q_{i_h}. \end{aligned}$$

Combining Paths

Let H_1 (resp. H_2) be closed half-plane bounded by straight line $p_m q_{i_h}$ and containing q_{i_1} (resp. q_{i_k}). Let P_0 be the vertex set of $\mathcal{P}$, and write

$$\begin{aligned} P_0 \cap H_1 &= \{p_{j_1}, p_{j_2}, \ldots, p_{j_s}\}, \\ &\quad \text{where } j_1 < j_2 < \cdots < j_s, \text{ and} \\ P_0 \cap H_2 &= \{p_{j'_1}, p_{j'_2}, \ldots, p_{j'_t}\}, \\ &\quad \text{where } j'_1 < j'_2 < \cdots < j'_t \end{aligned}$$

(note that we have $p_{j_s} = p_{j'_t} = p_m$). Then the paths $\mathcal{P}_1 = p_{j_1} p_{j_2} \ldots p_{j_s}$ and $\mathcal{P}_2 = p_{j'_1} p_{j'_2} \ldots p_{j'_t}$ are non-crossing monotonic paths in H_1 and H_2, respectively (Figure 4).

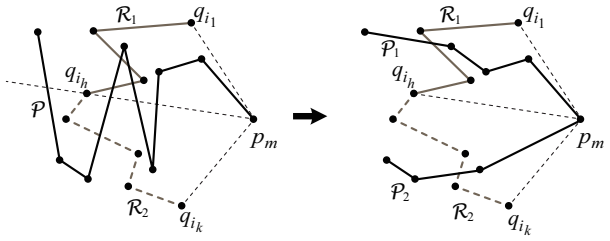


Figure 4

Case 1. $\mathcal{R}_1$ is increasing and $\mathcal{R}_2$ is decreasing:
In this case, we combine paths $\mathcal{P}_1$, $p_{j_s} q_{i_k}$ and ${\mathcal{R}_2}^{-1}$ to form a non-crossing monotonic path $\mathcal{S}_1$, and combine paths $\mathcal{P}_2$, $p_{j'_t} q_{i_1}$ and $\mathcal{R}_1$ to form another non-crossing monotonic path $\mathcal{S}_2$:

$$\begin{aligned} \mathcal{S}_1 &= p_{j_1} p_{j_2} \ldots p_{j_s} q_{i_k} q_{i_{k-1}} \ldots q_{i_h} \quad \text{and} \\ \mathcal{S}_2 &= p_{j'_1} p_{j'_2} \ldots p_{j'_t} q_{i_1} q_{i_2} \ldots q_{i_h}. \end{aligned}$$

Since

$$\begin{aligned} &(\text{the length of } \mathcal{S}_1) + (\text{the length of } \mathcal{S}_2) \\ &= [(\text{the length of } \mathcal{P}) + 1] \\ &\qquad +[(\text{the length of } \mathcal{Q}) + 1] \\ &= (a_m + 1) + (k+1) \\ &> d(\sqrt{n}-1) + \frac{1}{d}\sqrt{n} + \frac{3}{2} \quad \text{(by (7) and (9))} \\ &> \left(d + \frac{1}{d}\right)(\sqrt{n}-1), \end{aligned}$$

at least one of $\mathcal{S}_1$ or $\mathcal{S}_2$ has length at least $\frac{1}{2}\left(d + \frac{1}{d}\right)(\sqrt{n}-1) = c(\sqrt{n}-1)$, as desired.

Case 2. $\mathcal{R}_1$ is decreasing and $\mathcal{R}_2$ is increasing:
In this case, we combine paths $\mathcal{P}_1$, $p_{j_s} q_{i_h}$ and $\mathcal{R}_2$ to form a non-crossing monotonic path $\mathcal{T}_1$, and combine paths $\mathcal{P}_2$, $p_{j'_t} q_{i_h}$ and ${\mathcal{R}_1}^{-1}$ to form another non-crossing monotonic path $\mathcal{T}_2$. The rest of the argument is quite similar to the argument in Case 1.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 24540144 and CONACyT(Mexico) grant CB-2012-01-0178379.

References

[1] F.R.K. Chung. *On unimodal subsequences.* J. Combin. Theory Ser. A **29** (1980) 267–279.

[2] J. Czyzowicz, E. Kranakis, D. Krizanc, J. Urrutia. *Maximal length common non-intersecting paths.* Proc. Eighth Canadian Conference on Computational Geometry, August 1996, Ottawa, 180–189.

[3] P. Erdős, G. Szekeres: *A combinatorial problem in geometry.* Compositio Math. **2** (1935) 463–470.

[4] T. Sakai, J. Urrutia. *Monotonic polygons and paths in weighted point sets.* Computational Geometry, Graphs and Applications (Lecture Notes in Computer Science **7033** (2011)) 164–175.

Optimal Straight-line Labels for Island Groups

Arthur van Goethem* Marc van Kreveld† Andreas Reimer‡ Maxim Rylov‡ Bettina Speckmann*

1 Introduction

Maps are used to solve a wide variety of tasks, ranging from navigation to analysis. Often, the quality of a map is directly related to the quality of its labelling. Consequently, a lot of research has focussed on the automatization of the labelling process (for an overview see [11]). Surprisingly the (automated) labelling of island groups has received little attention so far. This is at least partially caused by the lack of cartographic principles. Though extensive guidelines for map labelling exist (e.g., [6, 12]), information on the labelling of groups of islands is surprisingly sparse.

We take an algorithmic approach and focus on optimal solutions to the problem using different settings. Comparing manual and optimal labels may give insight in subconscious rules applied by cartographers.

Contribution. We define a formal framework for island labelling. The framework spawns a large series of unexplored computational geometry problems. In this paper we start by looking at a straight-line label. In Section 2 we describe two algorithms for a straight-line label that is, or is not, allowed overlap with islands. Section 3 discusses several extensions to these algorithms for closely related problems.

Framework. We assume the input to the island labeling problem is a set S of k islands, given as simple polygons P_1, ..., P_k, with n vertices in total. We abstract away from the label itself and assume the label can be represented by its placement area. To this end we assume the label has a baseline that is either straight, circular, or a Bézier curve. The placement area is defined by perpendicularly extruding the baseline by a (possibly zero) height. Besides the shape, labels may be optimal with respect to three characteristics. Firstly, the distance to the islands may be computed using different points of measure. Specifically, these include the centroid of the polygon, the point on the polygon closest to the label, and the complete area of the polygon. Secondly, different distance measures may be used. Three distance measures used in cartography are included, minimizing: the maximum distance (*min-max*), the sum of distances (*min-sum*), the sum of squared distances (*min-sum-sq*). Lastly, we (dis)allow overlap of the label with the islands. For all labels we make the assumption that the label is long enough to measure distance perpendicularly.

In this paper we focus on straight-line labels optimizing the min-max distance to the closest point of each island. We are interested in strictly optimal labels to prevent subjectivity. For practical labelling purposes a trivial discretization may be sufficient.

Related work. Many algorithms have been developed for labelling maps [11]. Labelling islands groups has received less attention, but was addressed by Van Kreveld and Slechter [7]. Their algorithm places a non-overlapping label in the position minimizing the maximum distance from the label to each island of the group. They, however, require labels to be horizontal and discretize the space of possible label placements.

Several known algorithms can directly be applied to labelling. All algorithms assume that we are dealing with a point set, which is true if we have a fixed point of measurement for each island. Furthermore, all algorithms apply to a straight-line label that may overlap islands. The rotating calipers algorithm [9] optimizes the min-max distance in $O(n \log n)$ time. Using known results from Dey [3], Brodal and Jacob [2] and Edelsbrunner and Welzl [4], the label optimizing the min-sum distance can be computed in $O(n + k^{4/3} \log k)$ time. Finally, Deming regression optimizes the min-sum-sq distance in $O(n)$ time. For circular arcs the minimum width annulus [1] solves the min-max distance problem in $O(n^2)$ time.

2 Optimal, straight-line label

To compute an optimal, straight-line label we make use of the arrangement of possible label positions in *dual space* [1]. We start with the simple case consisting of a straight-line label that is allowed overlap with the islands. We focus on the min-max distance measure to the closest point of each island. An island intersecting the label has distance 0. For now we assume labels have no height and, hence, are represented by a single line. As stated before, we assume we have a set S of k islands, given as simple polygons P_1, ..., P_k, with n vertices in total. We also assume that all labels are sufficiently long to measure all distances perpendicular.

*Dept. of Mathem. and Computer Science, TU Eindhoven, The Netherlands, [a.i.v.goethem|b.speckmann]@tue.nl.
Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.102 (AvG) and no. 639.023.208 (BS).
†Dept. of Information and Computer Sciences, Utrecht University, The Netherlands, m.j.vankreveld@uu.nl
‡Institute of Geography, Heidelberg University, Germany, [andreas.reimer|maxim.rylov]@geog.uni-heidelberg.de

Overlapping label. As we are only interested in the distance to the closest point, we first compute the convex hull of each island in $O(n)$ time [8] and replace each island by its convex hull. The k convex polygons in primal space, become k funnels in dual space. The *top and bottom boundaries* of these funnels are x-monotone polylines. Between each pair of islands there exist at most four tangents. Consequently, the top and bottom of any pair of funnels intersect each other at most four times. The boundaries of the funnels form a set of $2k$ x-monotone, 2-intersecting polylines with $O(n)$ vertices total.

For any fixed rotation, the furthest vertex (or edge) below the label in primal space, is on the upper envelope of the lower boundaries in dual space. As all the funnel-boundaries are 2-intersecting, we can compute the upper envelope of the lower boundaries in $O(n + k \log k)$ time and it has complexity $O(n)$. A similar argument holds for the furthest vertex above and the lower envelope of the upper boundaries.

Lemma 1 *The optimal solution for a fixed rotation is halfway between the upper- and lower-envelope.*

The optimal solutions for each rotation are located on a polygonal line that is exactly centered between the upper- and lower- envelope, and it has $O(n)$ complexity. When an optimal solution intersect all islands it is not required to have equal distances to either side. The solution having equal distances to the closest vertex (edge) on either side, however, is also a valid optimal solution.

Lemma 2 *For each segment s of the solution-line we can compute the optimal position in $O(1)$ time.*

Proof. Let x_{start} be the x-coordinate of the start of s. Let d_{start} be the vertical distance to the upper- (or lower-) envelope at x_{start}. While we move along s the distance to the upper envelope changes by a linear factor c_1 in x. For a shift of δ along the x-axis, the vertical distance in dual space is $f_s(\delta) = d_{start} + c_1 * \delta$. The distance to the closest point in primal space is

$$g_s(\delta) = \frac{(d_{start} + c_1 * \delta)^2}{(x_{start} + \delta)^2 + 1} \qquad (1)$$

The optimal position is at the minimum over the domain given by segment s. This minimum can be at an endpoint of s or in the middle. □

Theorem 3 *We can compute the optimal straight-line label minimizing the maximum minimum distance over all islands in $O(n + k \log k)$ time.*

Intersection-free label. Positions exactly halfway between the upper- and lower-envelope may result in labels that overlap one or more islands. For intersection-free labels we require more information. An optimal intersection-free label need not necessarily have an equal distance to the furthest closest point on either side. Therefore, we compute the complete arrangement of all $2k$ boundaries. In this arrangement we find the optimal intersection-free solution in $O(nk + k^2 \log k)$ time.

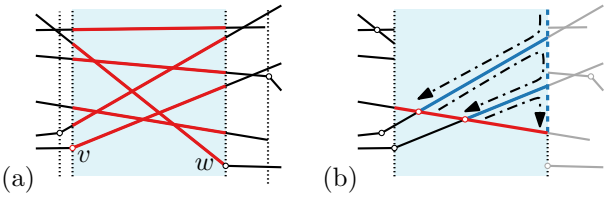


Figure 1: (a) The section of the arrangement (red) between vertices v and w, with $C = 7$ intersections. (b) The number of edges traversed to insert a new line with c intersections is $O(c)$.

Lemma 4 *The arrangement formed by all $2k$ boundaries can be computed in $O(nk)$ time.*

Proof. Make a single sorted list V of all vertices (excluding intersections), sorted along the x-axis, in $O(n \log k)$ time by merging the vertex-lists of the $2k$ boundaries. Sort all boundaries by decreasing slope of their first segment in $O(k \log k)$ time.

We compute the *complete arrangement* by performing a sweep along the x-axis. Define a *section* (v, w) of the complete arrangement as the arrangement consisting of all vertices u with $v_x \leq u_x < w_x$ and all maximal segments of edges having endpoints s, t, such that $v_x \leq s_x < w_x$ and $v_x \leq t_x < w_x$ (see Fig. 1(a)).

Let v_i, w_{i+1} be two consecutive vertices in V having index i and $i + 1$ respectively. In step i of the sweep we compute section (v_i, w_{i+1}) of the complete arrangement. The start and end of the arrangement are included by adding dummy vertices to V at minus infinity and infinity. Each section is computed in $O(k+C)$ time, where C is the number of intersections in this section.

As each step is between consecutive vertices in V, within each section all boundaries form straight lines. Within a section (v, w) we introduce all boundary-lines in sorted order (being sorted bottom to top at v_x). This order is known at the start of the algorithm and maintained during the sweep.

When introducing a line we find all intersections with the previously introduced lines. To find the intersections we move in a clockwise fashion through the arrangement built so far (see Fig. 1(b)). Any traversed line is either a boundary introduced earlier (starting lower) and ending higher, or the "virtual line" at w_x. Hence, the number of distinct lines we traverse is at most $O(c)$, where c is the number of intersections we detect by introducing this line. As lines

are 1-intersecting, the traversed edges in the worst case form a Davenport-Schinzel-sequence [10] of order 2 using c symbols, having complexity $O(c)$. Thus, when introducing a single boundary-line we spend at most $O(c)$ time.

The total number of intersections in a section is C. We may spend an extra constant amount of work per boundary, so introducing all lines takes $O(k + C)$ time. While introducing the lines we use insertion sort to obtain the sorted order at the end of each section in $O(k + C)$ time as well.

Because the number of intersections in the complete arrangement is bounded by $O(k^2)$, the algorithm runs in $O(nk + k^2) = O(nk)$ time. □

During creation of the arrangement we also keep track of the left-most point of each face. This point is uniquely defined as all lines are strictly x-monotone. The non-closed faces on the left of the arrangement maintain pointers to both infinite rays. We also store for each face if it is covered by a funnel. Faces covered by a funnel are defined to be *illegal*. An edge of the arrangement separating two illegal faces is also considered *illegal*, all other edges are *legal*.

Lemma 5 *The centerline may cross the arrangement $O(nk)$ times.*

Lemma 6 *The centerline can be inserted in the arrangement in $O(nk)$ time.*

If, for a fixed rotation, the point on the centerline is illegal, then clearly the closest legal point with the same rotation is optimal. The closest legal points form subedges on the edges of the arrangement (see Fig. 2).

Lemma 7 *We can find the closest legal subedges above the centerline in $O(nk + k^2 \log k)$ time.*

Proof. Remove from the computed arrangement all edges that are illegal in $O(n + k^2)$ time. Let a *chain* be a maximal series of consecutive edges connected by degree two vertices. The resulting arrangement has, excluding the centerline, $O(k^2)$ non-intersecting chains with $O(n+k^2)$ vertices total. We merge all vertices of the chains into a sorted list in $O((n+k^2) \log k)$ time. Now we merge the sorted list with the sorted vertices of the centerline in $O(nk)$ time to get a sorted list of all possible events.

We create a red-black tree (RB-tree) of all chains starting at the left-most end of the arrangement sorted by vertical order in $O(k \log k)$ time. The centerline is also added to the RB-tree. To all leafs of the RB-tree we add pointers to the neighboring leafs so that we can query for neighbors in $O(1)$ time.

We now move a sweepline over all vertices using the sorted list. Two types of events may occur. Firstly, when the first vertex of a new chain is reached, we add the chain to the red-black tree in $O(\log k)$ time. Similarly, when the last vertex of a chain is reached, we remove it. Secondly, when a chain intersects the centerline, we switch their positions in the red-black tree in $O(1)$ time.

At each event we can in $O(1)$ time determine the closest legal chain above the centerline. This may possibly be the centerline itself if it currently legal. Hence, we can trace the subedges above the centerline in $O(nk + k^2 \log k)$ time. □

The closest point to the centerline may be on the closest legal subedge above or below the centerline or may be on the centerline itself. However, the number of times where the point switches between these lines is bounded by $O(nk)$.

Lemma 8 *The closest legal line can switch at most $O(nk)$ times between the lower and upper legal line.*

Proof. Let W be the ordered set of vertices of the complete arrangement (including centerline) by x-coordinate. Let v and w be two consecutive vertices in W and $v_x \neq w_x$. We define a *slab* as the set of maximal segments of edges having endpoints s, t with $s_x = v_x$ and $t_x = w_x$ (see Fig. 3). There are $O(nk)$ vertices in the arrangement, so also $O(nk)$ slabs. For each slab all the segments are straight and non-intersecting. The closest legal point can switch at most once per slab and, thus, $O(nk)$ total. □

We can easily find the points on the centerline where the closest legal line changes and insert an extra vertex on the centerline. For each new edge of the centerline, the distance to the furthest closest point

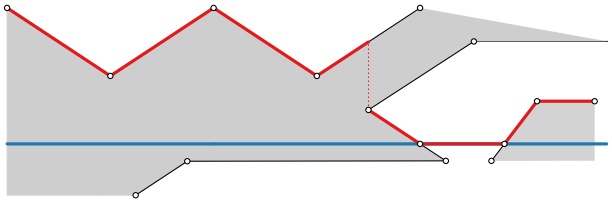


Figure 2: Centerline (blue), illegal faces (grey) and the closest legal subedges (red).

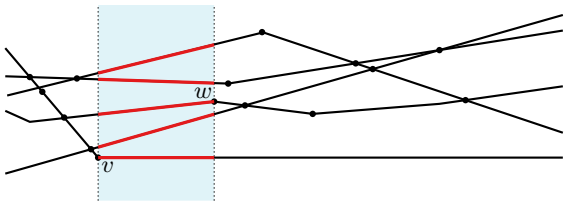


Figure 3: A *slab* of the complete arrangement (blue) and the belonging segments of edges (red). There are $O(nk)$ slabs in the complete arrangement.

changes linearly by c_1 and the distance to the closest legal point also changes linearly by c_2. Hence, similar to the overlapping version, we can create a closed formula for each segment s based on the shift δ along the x-axis.

$$g_s(\delta) = \frac{(d_{start} + o_{start} + (c_1 + c_2) * \delta)^2}{(x_{start} + \delta)^2 + 1} \qquad (2)$$

Theorem 9 *We can compute the optimal straight-line label optimizing the min-max distance that is non-overlapping in $O(nk + k^2 \log k)$ time.*

3 Extensions

In the previous section we have given algorithms for a (non-)overlapping label using the min-max distance. The basic idea can also be applied to solve different problems. In this section we give two examples.

Minimizing the sum of distances. It is easy to show that an optimal solution has equally many islands placed to either side of the label (and may possibly intersect some others). Hence, for a fixed rotation the optimal (possibly overlapping) solution can easily be found in $O(n + k \log k)$ time. For an intersection-free label in a fixed rotation, the optimal placement is the legal placement closest to the above solution.

In both cases it holds that if an optimal solution exists not tangent to any island, then there is also an optimal solution that is. Thus, the optimal solution in dual space is located on the edges of the arrangement.

We can compute the arrangement in $O((n + k^2) * \log k)$ time using a sweep line. While sweeping the arrangement we keep track of the faces having an equal number of islands above and below it. We compute the summed distance at the start and its linear change in the first slab in $O(k)$ time. For each event we update the linear change and the summed distance at the start in $O(1)$ time. We can compute the optimal position per edge in $O(1)$ time. Finding the closest legal edge can be done in $O(\log k)$ time as well. Hence, in $O((n+k^2) \log k)$ time we can find the optimal non-overlapping label.

Non-zero-height labels. Zero-height labels are not useful for real labelling. For an overlapping label, the optimal non-zero-height label is always centered on the optimal zero-height solution.

If we wish to place an intersection-free label of height h, this is equivalent to offsetting all islands by $h/2$ and computing a zero-height label. The vertices of the islands in primal space, however, become circular arcs. Consequently, in dual space we do not have polylines of line-segments, but of curves. A vertex $p = (p_x, p_y)$ of the top side of an islands being offset by a distance $h/2$ becomes a curve $p^* = p_x * x - p_y - \sqrt{x^2 + 1} * h/2$ in dual space. Vertices on the bottom side of islands become curves $p^* = p_x * x - p_y + \sqrt{x^2 + 1} * h/2$.

We can still compute the arrangement, the centerline, and the closest legal segments in $O(nk+k^2 \log k)$ time. For each segment we can in $O(1)$ time compute the optimal position, so we can find the global optimum in $O(nk+k^2 \log k)$ time. A similar argument can be made for the min-sum distance measure resulting in an $O((n + k^2) \log k)$ time algorithm.

Future work. In this paper we gave an $O(nk)$ algorithm to compute the arrangement. Technically, this is not necessary as a sweep-line algorithm can obtain the same $O(nk + k^2 \log k)$ time-bounds for the complete algorithm. However, the entire first section of our algorithm runs in $O(nk)$ time. In the worst case the centerline may intersect the arrangement at most $O(nk)$. This bound is also achievable. The final extra log-factor in $O(k^2 \log k)$ only turns up once. In future work we might investigate if this log-factor can also be removed, reaching the $O(nk)$ runtime. Current investigations, however, indicate a possible relation to the Sorting-(X+Y) problem [5].

References

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry, 3rd edition.* Springer, 2008.

[2] G. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd Annual IEEE Symp. on Foundations of Computer Science*, pages 617–626. ACM, 2002.

[3] T. Dey. Improved Bounds for Planar k-Sets and Related Problems. *Discrete Comput. Geom.*, 19(3):373–382, 1998.

[4] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrange-ments with applications. *SIAM J. on Computing*, 15(1):271–284, 1986.

[5] M. Fredman. How good is the information theory bound in sorting? *Theoret. Comput. Sci.*, 1:355–361, 1976.

[6] E. Imhof. Positioning names on maps. *American Cartographer*, 2(2):128–144, 1975.

[7] M. van Kreveld and T. Slechter. Automated label placement for groups of islands. *Proc. 22th Int. Cart. Conf.*, 2005.

[8] A. Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25(1):11–12, 1987.

[9] M. Shamos. Computational geometry, 1978. Ph.D. thesis. Dept. of CS, Yale Univ.

[10] C. Toth, J. O'Rourke, and J. Goodman. *Handbook of Discrete and Computational Geometry.* CRC press, 2004.

[11] A. Wolff and T. Strijk. A map labeling bibliography, 2009. http://i11www.iti.uni-karlsruhe.de/ awolff/map-labeling/bibliography/.

[12] C. Wood. Descriptive and Illustrated Guide for Type Placement on Small Scale Maps. *Carto. J.*, 37(1):5–18, 2000.

Clustered Edge Routing*

Quirijn W. Bouts† Bettina Speckmann†

1 Introduction

Graphs are an important tool to express relational data. The classic method to depict graphs is a node-link diagram where vertices (nodes) are associated with each object and edges (links) connect related objects. Node-link diagrams represent graphs in the most direct way. However, they quickly appear cluttered and unclear, even for moderately sized graphs. If the positions of the nodes are fixed – because they represent geo-referenced data or are laid out according to functional requirements – then suitable link routing is the only option to reduce clutter.

We present a novel link clustering and routing algorithm which respects (and if desired refines) user-defined clusters on the links. Our input is a node-link diagram with fixed node positions and optionally a user-defined clustering on the links and/or a set of disjoint polygonal obstacles. Our clustering method is based on a *well-separated pair decomposition* (WSPD) and we route link clusters individually on a *sparse visibility spanner*. To completely avoid ambiguity we draw each individual link and guarantee that clustered links follow the same path in the routing graph. (We call the edges of the routing graph 'edges' and use 'links' for the input from the node-link diagram.) Our algorithm also ensures that clusters are not drawn close to nodes and do not cross obstacles (see Fig. 1).

Holten and van Wijk [11] formalized four *edge compatibility measures* which indicate how similar links are. In Section 2 we argue that the clusters induced by a WSPD consist of compatible links according to the measures of Holten and van Wijk. Users can hence simply vary the separation constant of the WSPD to find a balance between few clusters of less compatible links and many clusters of very compatible links.

In Section 3 we show how to route the link clusters defined by the WSPD along the greedy sparsification of the visibility graph. On the complete graph a greedy sparsification has several desirable properties, such as a provable angle constraint as well as low total weight. Under realistic input assumptions we can prove that these properties still hold for the sparsification of the visibility graph. In Section 4 we describe our complete routing algorithm, including preprocessing, link ordering, and crossing minimization.

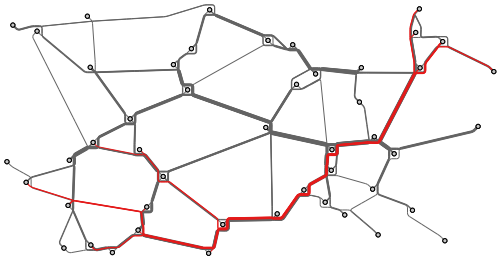


Figure 1: Clustered edge routing, one cluster in red.

Related work. Our work is closely related to the approach by Pupyrev *et al.* [2, 16] who use a routing graph based on a Yao-spanner of the visibility graph. The edges of a Yao-spanner can get arbitrarily close and angles can be arbitrarily small. Pupyrev *et al.* hence go through an extensive and computationally expensive iterative optimization step to improve it. Dwyer and Nachmanson [9] also use visibility graphs, albeit approximate ones, to route edges. They describe two approaches. The first uses a spatial decomposition and requires node movement. It can hence not be used to draw graphs with fixed node positions. The second approach also uses the Yao-spanner. All these techniques route links by using shortest paths on the routing graph. Link clustering is hence simply induced by the routing graph and does not necessarily satisfy any similarity measures. Furthermore, neither of these methods supports user-defined link clusters.

Dwyer et al. [8] integrate link routing techniques into a force-directed layout. Their method requires a feasible initial routing and moves vertices.

Various techniques reduce link clutter by *bundling* links which are "close". Gansner et al. [10] use a circular graph layout and route links either on the inside or the outside of the circle. Holten and van Wijk [11] describe a force-directed approach and use the aforementioned compatibility measures to determine the strengths of forces. Cui *et al.* [6] propose a geometry-based approach which uses a control mesh. Lambert *et al.* [13] use a combination of the Voronoi diagram and a quadtree as a multi-resolution grid for routing links. Bundling methods generally draw bundled links on top of each other. Hence it can be difficult to decide unambiguously if two nodes are connected. Luo *et al.* [14] propose a method which is ambiguity-free, but bundled links need to share a common node.

*A full version of this abstract will appear in Proc. 8th IEEE Pacific Visualization Symposium, 2015.

†Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands, [q.w.bouts|b.speckmann]@tue.nl. Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208.

2 Clustering links via a WSPD

The well-separated pair decomposition (WSPD) was introduced by Callahan et al. [5]. Two point sets A and B are well-separated if they can be enclosed in two circles of equal diameter which are "far apart" relative to their diameter. More precisely, point sets A and B, with bounding boxes $R(A)$ and $R(B)$, are said to be s-well-separated for some separation constant $s > 0$ if $R(A)$ and $R(B)$ can be enclosed in two disjoint equal diameter circles C_A and C_B and the distance between C_A and C_B is at least s times the diameter of C_A. The WSPD of a set of points P with separation constant s is a sequence of m pairs $\{A_i, B_i\}$ of nonempty subsets of P such that

1. for each $1 \leq i \leq m$, A_i and B_i are well-separated with respect to s.
2. for any two distinct points p and q there is exactly one pair (A_i, B_i) such that p is in one set and q in the other.

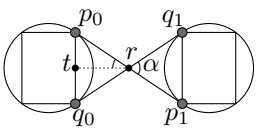
The number of well-separated pairs m is also called the size of the WSPD. If the separation constant s is indeed constant we can compute a WSPD of size $O(n)$ on a point set in the plane in $O(n \log n)$ time, see [15] Chapter 9. Every well-separated pair (A_i, B_i) induces a link cluster: each link with one endpoint in A_i and the other endpoint in B_i is part of the cluster.

Link compatibility measures. Holten and van Wijk introduced four measures which concern the angle, scale, and position of a pair of links, as well as the visibility between them. Consider the s-well-separated pair $\{A, B\}$. We examine the compatibility measures on any two links $e = (p_0, p_1)$ and $f = (q_0, q_1)$ with $p_0, q_0 \in A$ and $p_1, q_1 \in B$. We assume that the link e is at most as long as the link f. We use D to denote the diameter of the circles C_A and C_B.

Angle compatibility. Links in the same cluster should have a similar angle. We define the angle α between two non-parallel links as the smallest angle between the lines induced by the links. The angle of parallel links is 0.

Lemma 1 *The angle α between e and f is bounded by $\alpha \leq 2 \cdot \tan^{-1}(\frac{1}{s})$ for $s \geq 1$.*

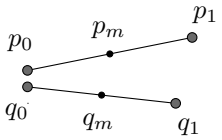
Proof. The figure shows a worst case configuration of e and f with respect to α. We have $|(p_0, t)| \leq 0.5D$ and $|(t, r)| \geq 0.5sD$ as rough bounds. We can now bound α by $2 \cdot \tan^{-1}(\frac{1}{s})$. □



Scale compatibility. Links in the same cluster should have similar length.

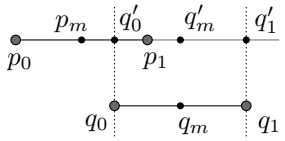
Lemma 2 *The difference in length of e and f is at most $2 \cdot D$. The length ratio of f to e is bounded by $\frac{|f|}{|e|} \leq \frac{s+2}{s}$.*

Position compatibility. Links which are close to each other should be more likely to end up in the same cluster. Holten and van Wijk measure "close to each other" by considering the distance between the midpoints p_m and q_m of links e and f in relation to the average link length of e and f.



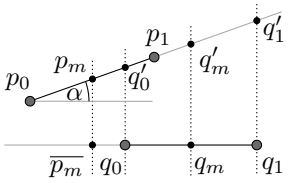
Lemma 3 *The difference in position of links e and f with midpoints p_m and q_m is $|(p_m, q_m)| \leq D$. The ratio of the difference in position to the average length is bounded by $\frac{|p_m q_m|}{(|e|+|f|)/2} \leq \frac{1}{s}$.*

Visibility compatibility. Let q'_m be the point on the line induced by e that when projected onto f coincides with its midpoint q_m. The visibility compatibility of e with f is defined by the normalized distance between the midpoint of e (p_m) and q'_m. To normalize this distance we divide by the length of the segment $q'_0 q'_1$ which when projected onto the line induced by f coincides with f.



Lemma 4 *Let q'_0, q'_1 and q'_m be the points on the line induced by e which, when projected onto f, coincide with q_0, q_1, and q_m. The visibility compatibility is bounded by $\frac{|(p_m, q'_m)|}{|(q'_0, q'_1)|} \leq \frac{1}{s}$ for $s > 1$.*

Proof. Let α be the angle between the lines induced by e and f which using Lemma 1 we can bound as $\alpha < 2 \tan^{-1}(\frac{1}{s})$. Let $\overline{p_m}$ be the projection of p_m onto the line induced by f. We have $|(\overline{p_m}, q_m)| \leq |(p_m, q_m)| \leq D$ by the triangle inequality. This implies $|(p_m, q'_m)| \leq \frac{D}{\cos(\alpha)}$. From the definition of s-well-separated we have $|(q_0, q_1)| \geq sD$, which implies that $|(q'_0, q'_1)| \geq \frac{sD}{\cos(\alpha)}$. We now have $\frac{|(p_m, q'_m)|}{|(q'_0, q'_1)|} \leq \frac{D}{\cos(\alpha)} \cdot \frac{\cos(\alpha)}{sD} = \frac{1}{s}$. □



Increasing the separation constant s of the WSPD improves all four compatibility measures. Users can hence vary s to find a balance between few clusters of less compatible links and many clusters of very compatible links. If the user has specified clusters we test if they are also spatially clustered. If this is not the case, that is, if the endpoints of the clustered links are not well-separated, we can refine the user-specified cluster into compatible clusters using the WSPD.

3 The routing graph

We add a small polygonal obstacle around each node, merge those obstacles which are "too close", and enclose merged obstacles within their convex hull. We use a combination of Voronoi and additional *in-cell* edges to connect nodes with their obstacle vertices.

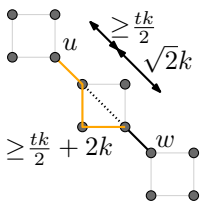
Let $d_G(p,q)$ denote the shortest-path distance between two vertices p and q in a graph $G = (V, E)$. A *geometric t-spanner* ($t > 1$) of G is a graph $G' = (V, E' \subseteq E)$ such that for any two vertices $p, q \in V$, $d_{G'}(p,q) \leq t \cdot d_G(p,q)$. The so-called *greedy spanner* is constructed by considering all edges $e = (p,q) \in E$ in non-decreasing order and adding them to E' if and only if $d_{G'}(p,q) > t \cdot d_G(p,q)$. For our routing graph we use the greedy spanner to sparsify the visibility graph. We also include all original obstacle edges.

The visibility graph can be computed in $O(n^2 \log n)$ time using a sweepline approach (see [7] Chapter 15). The greedy spanner can be computed in $O(n^2 \log n)$ time [3]. We use the $O(n^2 \log^2 n)$ algorithm of Bouts *et al.* [4] which is faster in practice.

The greedy sparsification of the complete graph is very sparse, it guarantees a lower bound for the angles between nearby edges, and it has a total weight of $O(MST)$, where MST denotes the weight of the minimum spanning tree ([15] Chapter 6, 14). A lower bound on the angle between adjacent edges is important when routing unambiguously. We sketch the argument for the greedy spanner and illustrate how (under realistic input assumptions) a similar proof holds for our routing graph.

Let $(v,u),(v,w)$ be two edges in the greedy spanner with angle α at v and let (v,u) be considered before (v,w) by the algorithm ($|vu| \leq |vw|$). For $\alpha < \frac{\pi}{4}$ we have that $|uw| < |vw|$ and hence there is a $t-$path between u and w when considering (u,v). Furthermore, we have $t > 1/(\cos\alpha - \sin\alpha)$. Hence, if α is too small in relation to t, we have $|vu| + t|uw| < t|vw|$ contradicting that (v,w) is in the greedy spanner and implying a lower bound on α.

We now assume square axis-aligned obstacles with sidelength k, which are separated by at least $\frac{1}{2}tk$. Hence the sparsification of the visibility graph must include all obstacle edges since no other vertices are sufficiently close to form a $t-$path between their vertices. Let $\delta(u,w)$ denote the length of the shortest path between u and w in the visibility graph. Because of our input assumptions $\delta(u,w)$ is at most a small constant factor larger than $|uw|$ (see figure). Hence the same argument as above applies, with slightly worse constants.



By adjusting the dilation t, the user can find a balance between a more direction preserving or a cleaner, more abstract routing graph.

4 Routing and ordering links

The routing should respect the clustering. Using the nearest pair of nodes (one from each set of the corresponding well-separated pair) we determine *merge points*. All links in a cluster are routed via their merge points to ensure that they share a common sub-path.

To avoid unambiguity we draw links individually along the routing graph edges. Inspired by [16] we use bi-arcs to allow them to smoothly change directions at vertices and to ensure that they cross at large angles. Clustered links are drawn together to form a *ribbon of links*. This leads to a two-level ordering problem. We order both ribbons and the links within them to minimize crossings. We can order the links optimally in linear time since they do not pass through nodes [16].

Because of the tree-like structure of the ribbons ordering them among each other is NP-complete. We prove this by reduction from the 1-sided crossing minimization problem [12]: Given a two-layered (bipartite) graph $G = (\{L_0, L_1\}, E)$ and an ordering x_0 of vertices on layer L_0, is there an ordering x_1 of L_1 such there are at most k crossings?

Definition 1 (Ribbon ordering problem) *Given the ribbons routed through vertex v. Is there an ordering along its edges with at most k crossings?*

Lemma 5 *Ribbon ordering is NP-Complete.*

Proof. Consider an arbitrary instance of the one-sided crossing minimization problem $G = (\{L_0, L_1\}, E)$ where L_0 is the fixed layer. We denote the fixed vertices with $\bar{v}_1 \dots \bar{v}_{|L_0|}$ and those in L_1 by $v_1 \dots v_{|L_1|}$. We construct an instance of the ribbon ordering problem as illustrated in Fig. 2. We add edges $e_1 \dots e_{|L_0|}$ on the left side corresponding to L_0 and one edge on the right side. We add ribbons $r_1 \dots r_{|L_1|}$ to this edge corresponding to L_1. For each edge in $(\bar{v}_i, v_j) \in E$ we add a link from ribbon r_j to e_i.

We can now solve the original instance by finding an order of ribbons on the right edge. Since ribbon ordering is clearly in NP this proves NP-completeness. □

This close relation to 1-sided crossing minimization allows many of its heuristics to be adapted to ribbon ordering. For our implementation we adapted the

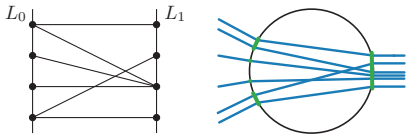
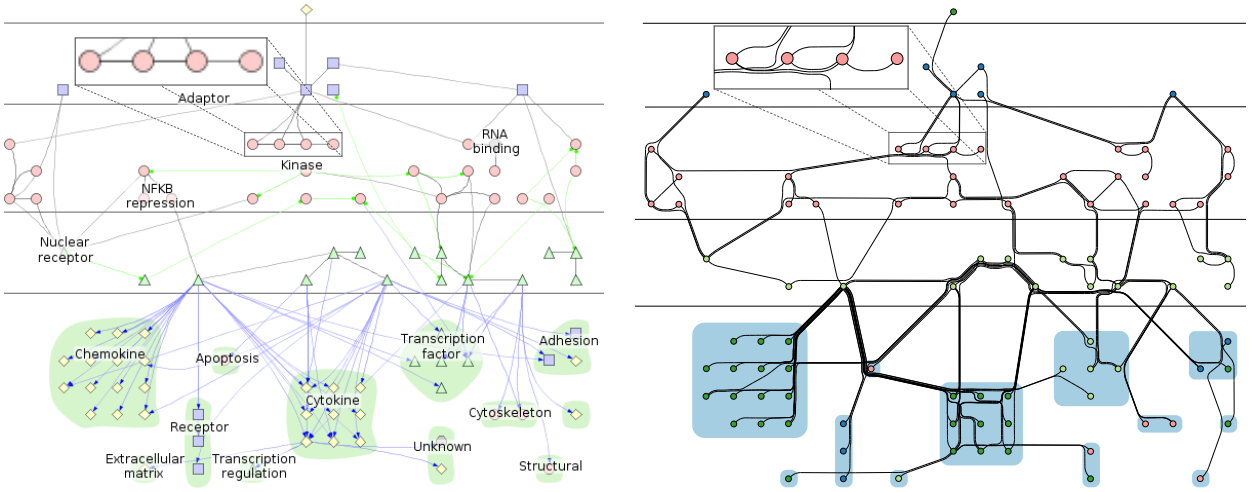


Figure 2: A 1-sided crossing minimization problem and the corresponding ribbon ordering instance. The four left edges act as a fixed layer.



(a) (b)

Figure 3: The TLR4 graph from [1] (91 nodes, 124 links), rendered by (a) Cerebral [1] and (b) our algorithm. Closeups of the same region show the ambiguity problems arising from overlapping links in the original drawing.

Barycenter heuristic [12] which works by calculating the "average ranking" for each ribbon.

We demonstrate our method on the TLR4 dataset which represents interactions between different biomolecules. Barsky *et al.* [1] used their Cerebral software to find node positions and used a simple spline based heuristic to draw links (see Fig. 3(a)). Fig. 3(b) shows the result of our algorithm on their node layout. Clusters were defined by experts and refined where needed using a separation of 1. The sparsification was computed for a dilation of $t = 1.8$.

Acknowledgements. The authors would like to thank Tim Dwyer for fruitful discussions and Tamara Munzner for the Cerebral use case.

References

[1] A. Barsky, T. Munzner, J. L. Gardy, and R. Kincaid. Cerebral: Visualizing Multiple Experimental Conditions on a Graph with Biological Context. *IEEE Trans. Visualization and Computer Graphics*, 14(6):1253–60, 2008.

[2] S. Bereg, A. E. Holroyd, L. Nachmanson, and S. Pupyrev. Edge routing with ordered bundles. `arXiv:1209.4227`.

[3] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid. Computing the greedy spanner in near-quadratic time. *Algorithmica*, 58(3):711–729, 2010.

[4] Q. W. Bouts, A. P. ten Brink, and K. Buchin. A framework for computing the greedy spanner. In *Proc. 30th Sympos. Computional Geometry*, pages 11–19. ACM, 2014.

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

[6] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Trans. Visualization and Computer Graphics*, 14(6):1277–84, 2008.

[7] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag, 3rd edition, 2008.

[8] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In *Proc. 14th International Symposium on Graph Drawing*, LNCS 4372, pages 8–19, 2007.

[9] T. Dwyer and L. Nachmanson. Fast edge-routing for large graphs. In *Proc. 17th International Symposium on Graph Drawing*, LNCS 5849, pages 147–158, 2010.

[10] E. R. Gansner and Y. Koren. Improved circular layouts. In *Proc. 14th International Symposium on Graph Drawing*, LNCS 4372, pages 386–398, 2007.

[11] D. Holten and J. J. Van Wijk. Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[12] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1(1):1–25, 1997.

[13] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862, 2010.

[14] S.-J. Luo, C.-L. Liu, B.-Y. Chen, and K.-L. Ma. Ambiguity-free edge-bundling for interactive graph visualization. *IEEE Trans. Visualization and Computer Graphics*, 18(5):810–821, 2012.

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

[16] S. Pupyrev, L. Nachmanson, S. Bereg, and A. Holroyd. Edge routing with ordered bundles. In *Proc. 19th International Symposium on Graph Drawing*, LNCS 7034, pages 136–147, 2012.

Mosaic Drawings and Cartograms

R. G. Cano* K. Buchin‡ T. Castermans‡ A. Pieterse‡ W. Sonke‡ B. Speckmann‡

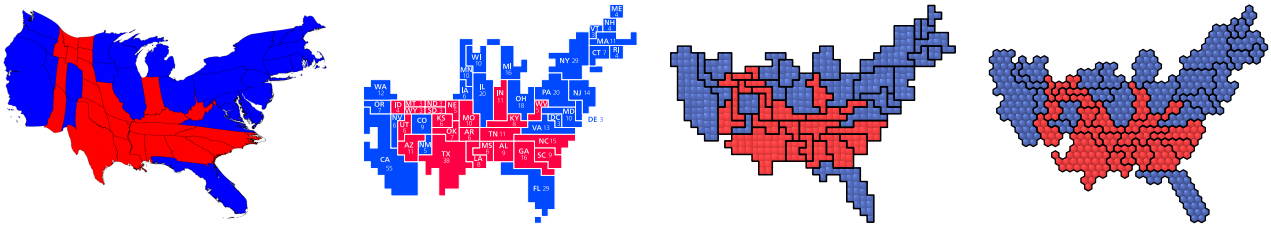


Figure 1: US election 2012, electorial college votes. Diffusion cartogram by M. Newman (Univ. Michigan), square mosaic cartogram (Wikipedia), square and hexagonal mosaic cartograms computed by our algorithm.

1 Introduction

Cartograms visualize quantitative data about a set of regions such as countries or states by scaling each region such that its area is proportional to its data value. There are several different types of cartograms and some algorithms to construct them automatically exist. The most common cartograms are *contiguous area cartograms* [8] (Fig. 1 left). Here the regions are deformed in such a way that adjacencies are kept. Contiguous area cartograms perform well if the data values are positively correlated to the land areas of the input regions, but producing good cartograms if this is not the case remains a challenge.

The size of regions in contiguous area cartograms is generally hard to judge. To remedy this situation several types of cartograms depict regions using simple geometric shapes like disks, squares or rectangles. When using rectangles, adjacencies and relative positions can be maintained [2, 10]. However, the rectangular shape is not very recognizable and hence Mumford et al. [4, 5] initiated the study of rectilinear cartograms where each region is represented by a rectilinear polygon. But the areas of general rectilinear polygons are again difficult to estimate and compare. This is much easier if the polygons are composed of a small number of unit squares (Fig. 1 middle two). We focus on this type of cartograms, or more generally cartograms which use multiples of simple tiles – usually squares or hexagons (Fig. 1 right) – to represent regions. In absence of a dedicated name in the literature we call such cartograms *mosaic cartograms*.

Mosaic cartograms using squares have been popularized by the New York Times, usually in the context of the US elections, but also to show changing demographics. Mosaic cartograms using hexagons are less frequent, examples are the "Indices of Deprivation 2010" by the Leicestershire County Council.

Mosaic cartograms communicate data well that consist of, or can be cast into, small integer units (for example, electorial college votes), they allow users to accurately compare regions, and they can often maintain a (schematized) version of the input regions' shapes. We propose the first method to construct mosaic cartograms fully automatically. To do so, we first introduce *mosaic drawings* of triangulated planar graphs. We then modify mosaic drawings into mosaic cartograms with zero cartographic error while maintaining correct adjacencies between regions.

Quality criteria. There are several quality criteria for (mosaic) cartograms. One of the most important ones is the *cartographic error*, which is defined for each region as $|A_c - A_s|/A_s$, where A_c is the area of the region in the cartogram and A_s is the specified area depending on the data value to be shown. In a mosaic cartogram a region is represented by an edge-connected set of tiles, which we call a *configuration*. Each configuration must be *simple*, that is, contain no holes. The *mosaic resolution* measures the (maximum and average) number of tiles used per region. We consider the following quality criteria in this paper:

- Average and maximum cartographic error
- Correct adjacencies of configurations: two configurations are adjacent if and only the corresponding input regions are adjacent
- Shape of the regions
- Relative positions of regions
- Mosaic resolution

It is generally challenging to simultaneously satisfy all criteria well. We decided to enforce correct adjacencies in our algorithm, that is, we produce only mosaic

*Institute of Computing, University of Campinas, Brazil, rgcano@ic.unicamp.br. Supported by FAPESP under projects 2012/00673-2 and 2013/23571-3.

‡Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands, [k.a.buchin|b.speckmann]@tue.nl, [t.h.a.castermans|a.pieterse|w.m.sonke]@student.tue.nl. K.B. and B.S. supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.106 and 639.023.208, respectively.

cartograms which have exactly the same configuration adjacencies as the corresponding regions of the input map. There is a clear trade-off between mosaic resolution and recognizability of the map. A low mosaic resolution, that is a small to medium number of tiles per region, allows users to explicitly count tiles and compare regions. A high mosaic resolution makes it easier to preserve shapes and relative positions and to achieve zero cartographic error. Fortunately our method can create mosaic cartograms with zero cartographic error for relatively low mosaic resolutions.

Most mosaic cartograms which are made by hand do neither preserve the adjacencies of all input regions nor their shapes. A common semi-automated approach is to take a map or a contiguous area cartogram and to overlay it with a suitable grid. This requires a rather high mosaic resolution, since otherwise rounding errors will easily destroy or create adjacencies. Furthermore, mosaic cartograms created in this way usually do not have zero cartographic error.

2 Mosaic Drawings

We define mosaic drawings for *plane triangulated graphs*, that is, planar graphs with a given embedding where every interior face is a triangle. Mosaic drawings are drawn on a *tiling* of the plane. Of particular interest are the uniform, and especially the regular tilings, although other types of tilings might also result in intriguing drawings. There are three types of regular tilings: the triangular, the square, and the hexagonal tiling. The triangular tiling uses two different rotations of the basic triangular shape and hence is visually a little more complex than the square or the hexagonal tiling.

We call a set of edge-connected tiles of a tiling $\mathcal{T}$ a *configuration*. We say that a configuration C is *simple* if its tiles are simply connected (C has no holes). Two configurations C_1 and C_2 are adjacent if and only if there is at least one tile $t_1 \in C_1$ and at least one tile $t_2 \in C_2$ such that t_1 and t_2 are edge-connected. A *mosaic drawing* $\mathcal{D}_\mathcal{T}(G)$ of plane triangulated graph $G = (V, E)$ on $\mathcal{T}$ represents every vertex $v \in V$ by a simple configuration $C(v)$ of edge-adjacent tiles from $\mathcal{T}$ in such a way that two vertices v and u of G are connected by an edge $e = (v, u)$ if and only if the configurations $C(v)$ and $C(u)$ are adjacent (see Fig. 2).

We say that a mosaic drawing is *simple* if the union of all configurations is simply connected, that is, the drawing has no holes. Mosaic drawings are a type of *contact representation*, since they do not draw edges explicitly, but imply them by the contact of the configurations.

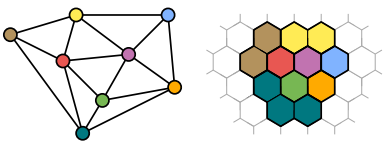


Figure 2: Simple mosaic drawing.

For which tilings do mosaic drawings exist? Below we show – with the help of results from the Graph Drawing and VLSI layout literature – that simple mosaic drawings exist for both square and hexagonal tilings. For a given (style of) mosaic drawings we can then consider various quality criteria, such as the size (number of tiles) of the drawing and the area (number of tiles inside a bounding box).

Square and hexagonal tilings. We show that any plane triangulation has a mosaic drawing on a square and hexagonal tiling via *orthogonal (grid) drawings* [6] of a graphs. Here the vertices are placed on an integer grid and the edges are routed along the grid.

Given a plane triangulation we can obtain a mosaic drawing in the following way as illustrated in Fig. 3: We first take the weak dual of the triangulation, which has a vertex for each triangle of the triangulation, and an edge for any pair of adjacent triangles. The dual of a triangulation has a maximum degree of three and can be laid out orthogonally on a grid. To obtain a mosaic drawing on a square tiling, we represent every grid point of the orthogonal drawing by four squares of the tiling and consistently distribute them to the configurations of the triangulation. On a hexagonal grid we can proceed in exactly the same way but shear the orthogonal drawing; alternatively we can directly embed the dual on a hexagonal grid [9].

The construction above shows that we can obtain mosaic drawings that are linear in the size of the corresponding orthogonal drawings. This allows us to derive complexity results for mosaic drawings from existing results for orthogonal drawings. Orthogonal drawings on small-area grids have been studied extensively [6, 12]. For instance, if the triangulation induces an outerplanar graph, we can obtain a mosaic drawing of relatively small area by making use of

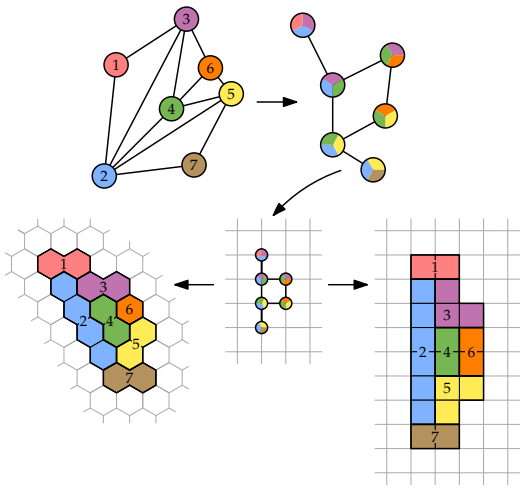


Figure 3: Transformation from orthogonal drawing to mosaic drawings via the (weak) dual.

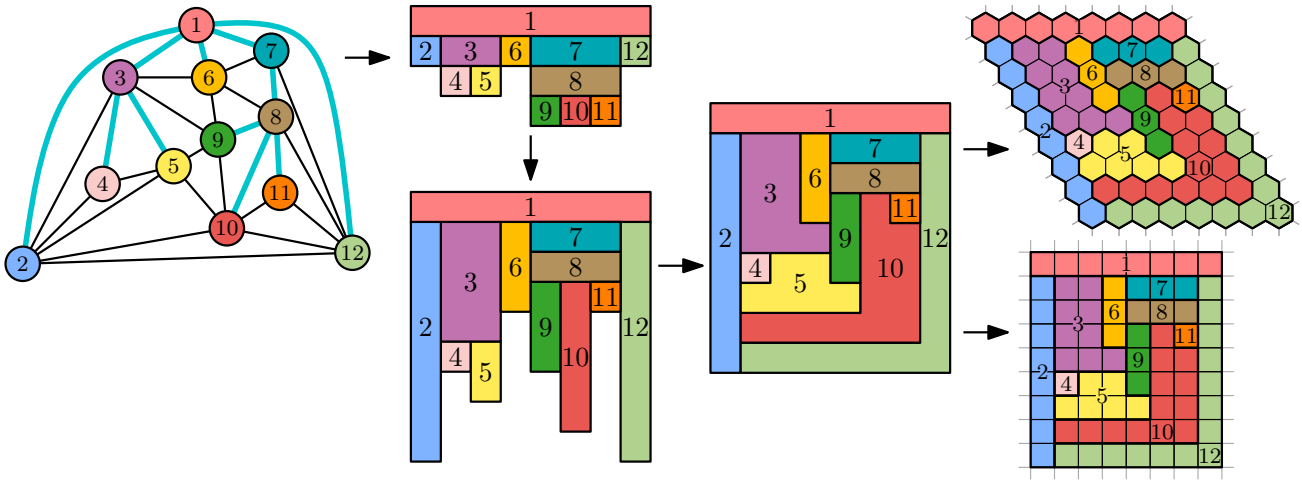


Figure 4: Constructing simple mosaic drawings for square and hexagonal tilings via an orderly spanning tree (inspired by [3]).

the fact that the dual is a binary tree [7]. However, minimizing the size (area, length, etc.) of an orthogonal drawing is NP-hard even if the orientations of the edges used are already given [1, 13], which makes it unlikely that we can efficiently minimize the area of a mosaic drawing.

3 Mosaic cartograms

Our input is now a triangulated planar graph $G = (V, E)$ together with integer weights $w(v)$ for each vertex $v \in V$. A mosaic cartogram for G is a simple mosaic drawing of G where each configuration $C(v)$ consists of exactly $w(v)$ tiles for each vertex $v \in V$. The construction of mosaic drawings from orthogonal drawings of the dual as sketched above does not necessarily produce *simple* mosaic drawings. While we could extend this construction and fill faces in a consistent way, we instead choose an alternative method which is based on *orderly spanning trees* [3] and produces compact simple mosaic drawings (see Fig. 4).

Orderly spanning trees are spanning trees with certain order relations between the nodes. Chiang et al. [3] show how to compute an orderly spanning tree ST for a planar triangulation G. They then construct a (vertical) visibility drawing for ST, which is stretched into a 2-visibility drawing of G. They grow horizontal branches to fill up gaps and finally shrink thick branches to height 1. The result is a square mosaic drawing of G. Since at most three regions meet at each intersection, we can directly shear the same drawing onto a hexagonal grid and so obtain a hexagonal mosaic drawing with the same complexity.

To compute mosaic cartograms, we start with a mosaic drawing of the (augmented) dual of the input map. Unfortunately the orderly spanning trees created by Chiang et al. [3] have a tendency to "curl inwards". The resulting mosaic drawings often do not retain any of the relative positions of the input nodes and are hence a rather poor starting point for mosaic cartograms (see Fig. 5 left). However, the spanning trees induced by a Schnyder labeling are also orderly spanning trees [11] and lead to mosaic drawings which do capture a significant part of the relative positions of the input (see Fig. 5 right top).

We use an iterative method to grow (or shrink) the configurations according to the input data. While doing so, we maintain the correct adjacencies at all times. We use so-called *guiding shapes* – configurations which represent the desired final state of each configuration $C(v)$ – to slowly nudge each configuration towards the correct number of tiles and the correct shape. Guiding shapes are iteratively moved using a force-directed approach to reduce overlap while moving the configurations into appropriate relative positions. Note that the configurations never overlap, only their guiding shapes might. We maintain a proper mosaic drawing at all times, although it might

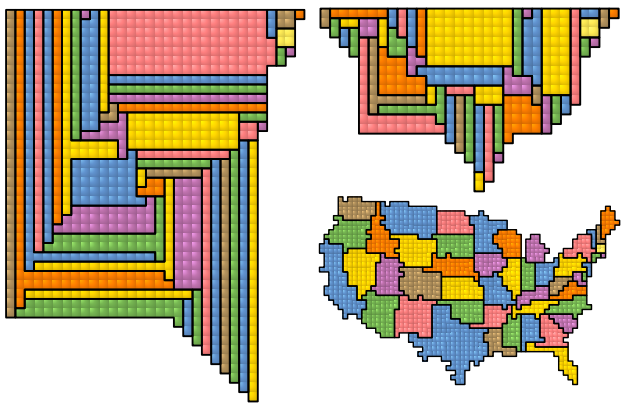


Figure 5: US: mosaic drawing based on Chiang et al. [3] (left) and on Schnyder labeling (top right), area cartogram with 1 square ≈ 5000 km^2.

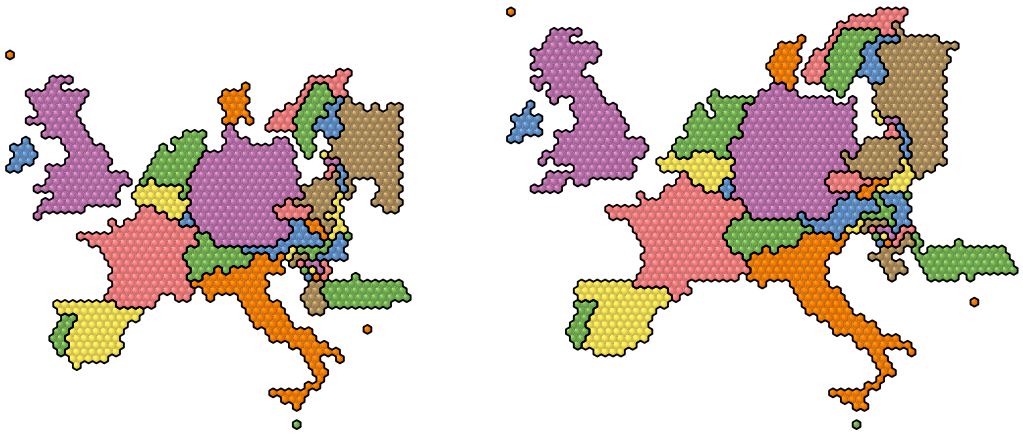


Figure 6: EU GDP in 2013: 1 hexagon per 20 billion USD, 1122 tiles, 1 incorrect hexagon (left), 1 hexagon per 15 billion USD, 1499 tiles, no error (right).

not always be simple. Each configuration, however, is always simple. This process stops as soon as no more progress can be made. Now most configurations have a good shape and the correct number of tiles, a few might have some tiles too many or too few. The final step of our algorithm uses a minimum cost flow formulation to assign the correct number of tiles to each configuration. This process maintains the correct adjacencies and disturbs the shapes as little as possible. It also removes any holes in the drawing which might have arisen during the iterative resizing and moving.

4 Future Work

An obvious direction for future work are other types of tilings, most notably the triangular tiling. Our method in principle extends to any uniform tiling, if one interprets it as a square or hexagonal tiling by grouping adjacent tiles appropriately (Fig. 7). However, more direct approaches and corresponding size and area bounds would be of interest.

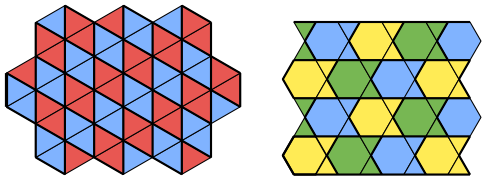


Figure 7: Interpreting the hexagonal tiling as a square tiling and the trihexagonal tiling as a hexagonal tiling.

References

[1] M. J. Bannister, D. Eppstein, and J. Simons. Inapproximability of orthogonal compaction. *J. Graph Algorithms and Applications*, 16(3):651–673, 2012.

[2] K. Buchin, B. Speckmann, and S. Verdonschot. Evolution strategies for optimizing rectangular cartograms. In *Proc. 6th Intern. Conference on Geogr. Information Science*, LNCS 7478, pages 29–42, 2012.

[3] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications. *SIAM Journal on Computing*, 34(4):924–945, 2005.

[4] M. de Berg, E. Mumford, and B. Speckmann. On rectilinear duals for vertex-weighted plane graphs. *Discrete Mathematics*, 309(7):1794–1812, 2009.

[5] M. de Berg, E. Mumford, and B. Speckmann. Optimal BSPs and rectilinear cartograms. *International Journal of Computational Geometry and Applications*, 20(2):203–222, 2010.

[6] C. A. Duncan and M. T. Goodrich. Planar orthogonal and polyline drawing algorithms. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, Chapter 7, pages 223–246. CRC Press, 2013.

[7] F. Frati. Straight-line orthogonal drawings of binary and ternary trees. In *Proc. 15th Intern. Conference on Graph Drawing*, LNCS 4875, pages 76–87, 2008.

[8] M. Gastner and M. Newman. Diffusion-based method for producing density-equalizing maps. *Proc. National Academy of Sciences of the USA*, 101(20):7499–7504, 2004.

[9] G. Kant. Hexagonal grid drawings. In *Graph-Theoretic Concepts in Computer Science*, LNCS 657, pages 263–276, 1993.

[10] M. v. Kreveld and B. Speckmann. On rectangular cartograms. *Computational Geometry: Theory and Applications*, 37(3):175–187, 2007.

[11] K. Miura, M. Azuma, and T. Nishizeki. Canonical decomposition, realizer, Schnyder labeling and orderly spanning trees of plane graphs. *Intern. Journal Found. of Computer Science*, 16(1):117–141, 2005.

[12] A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry: Theory and Applications*, 9(12):83—110, 1998.

[13] M. Patrignani. On the complexity of orthogonal compaction. *Computational Geometry: Theory and Applications*, 19(1):47–67, 2001.

Long Paths in Line Arrangements

Udo Hoffmann*† Linda Kleist* Tillmann Miltzow ‡

Abstract

An arrangement of lines partitions the plane into vertices, edges and faces. A (dual) path in a line arrangement is a sequence of faces where every two consecutive faces share an edge and each face occurs at most once. We prove that every arrangement of n lines has a path of length $n^2/3 - O(n)$. This is tight up to the linear order term.

We also consider arrangements of red and blue lines, where our paths must cross red and blue edges alternatingly: We describe an example of a line arrangement with $3k$ blue and $2k$ red lines with no alternating path longer than $14k$. Moreover, we show that any arrangement with n lines has a coloring such that there exist an alternating path of length $\Omega\left(n^2/\log n\right)$.

1 Introduction

A line arrangement $\mathcal{A}$ is a set of n lines in the Euclidean plane. The set of lines partitions the plane into vertices, edges, and faces. A line arrangement is called *simple* if no two lines are parallel and every point of the Euclidean plane is covered by at most 2 lines. A simple arrangement has $\binom{n}{2} + n + 1$ faces. A *(dual) path* in a line arrangement is a sequence of faces such that consecutive faces share an edge and no face appears more than once. The length of a path is defined as the number of involved faces. We are interested in bounds on the minimum length of a longest dual path in arrangements with n lines, i.e., in the function

$$f(n) := \min_{\mathcal{A}\in\mathbf{A}_n} \max_{P\in\mathcal{A}} |P|$$

where P is a dual path in $\mathcal{A}$, $|P|$ its length, and $\mathbf{A}_n$ the set of simple arrangements with n lines.

Aichholzer *et al.* [1] introduce this problem and show that each simple arrangement of n lines admits a path of length $n^2/4 - O(n)$. As an upper bound, they show that there are arrangements with n lines where the longest path is of length $n^2/3 + O(n)$.

Our main result is a (up to lower order terms) tight lower bound on the length of a longest path.

*{kleist,uhoffman}@math.tu-berlin.de, Department of Mathematics, Technische Universität Berlin

†supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

‡t.m@fu-berlin.de, Department of Computer Science, Freie Universität Berlin

Theorem 1 *In a simple arrangement of n lines, there exist a path of length $1/3n^2 - O(n)$.*

There also exist a colored version of the problem: The lines are colored (with red and blue), and a dual path has to be *alternating*, that is it does not traverse two edges of the same color consecutively. For this variant, Aichholzer *et al.* [1] show that there exists an alternating dual path between any two bicolored faces. This result implies the existence of an alternating path of length $\Omega(n)$ in any bicolored arrangement with n lines. This bound is tight up to a constant factor for any arrangement with $n-1$ red and 1 blue line. They ask if it remains true in the case of more balanced color classes. We extend the upper bound to a more balanced scenario.

Theorem 2 *There exists a simple arrangement of $3k$ red and $2k$ blue lines where any alternating dual path goes through at most $14k$ faces, for every odd k.*

This negative result led us to the question if for any line arrangement there exist a coloring allowing for a 'long' alternating path. Our answer is a bound for a randomly colored arrangement.

Theorem 3 *In a random bicoloring of an arrangement of n lines, there exists an alternating path of length $\Omega\left(n^2/\log n\right)$ with high probability.*

2 Long Paths in Line Arrangements

Here we give a simplified proof of the theorem from [1], namely that any line arrangement with n lines has a path of length $n^2/4 - O(n)$. Afterwards, we give the general proof idea of Theorem 1. Pick an unbounded face l_0, and consider a wiring diagram of $\mathcal{A}$ such that

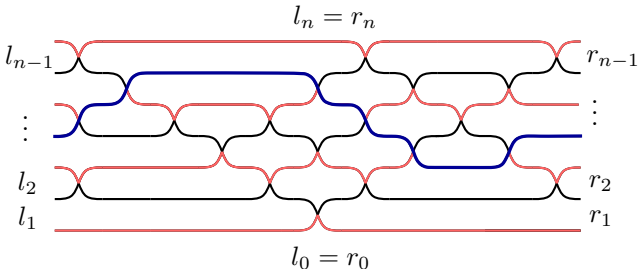


Figure 1: A wiring diagram with walls displayed in red; tunnels are between the walls.

l_0 is the bottom unbounded face. A wiring diagram is a standard way to represent the combinatorics of a line arrangement, see Figure 1 for an illustration; for a formal definition we refer to [4] or [6, Section 5.1].

The set of faces with a shortest path of length $i+1$ to l_0 is called *face level* L_i. There are exactly two unbounded faces in each level L_i, $1 < i < n$; we denote the left one by l_i, and the right one by r_i, see Figure 1. For $i \in \{0, n\}$, L_i consists of exactly one face; we also refer to them as the *top* and *bottom* face, respectively. We group pairs of adjacent face levels to tunnels, i.e. *tunnel* T_i is the union of the two face levels L_{2i-1} and L_{2i} for $1 \le i \le \lfloor n/2 \rfloor$.

As the dual graph induced by the faces of a tunnel is connected, we define an ordered set of tunnel paths $\mathcal{P} := \{P_i \mid i \in \{1, \ldots, \lfloor n/2 \rfloor\}\}$, with

$$P_i := \text{a path in tunnel } T_i \begin{cases} \text{from } l_{2i-1} \text{ to } r_{2i} & i \text{ odd}, \\ \text{from } r_{2i-1} \text{ to } l_{2i} & i \text{ even}. \end{cases}$$

We think of path P_i as oriented from the start to the end vertex. The path family $\mathcal{P}$ has the property of being *glueable*, that is the paths are pairwisely disjoint and the end faces of P_i and the start face of P_{i+1} are adjacent. Hence, the paths in $\mathcal{P}$ can be 'glued' to one path by concatenation: $P := P_1\, P_2\, P_3\, P_4\, \ldots$

Lemma 4 *Let Q be a path from l_i to r_j in $\mathcal{A}$.*
If $i, j \le n/2$, then $|Q| \ge i + j + 1$.
If $i, j \ge n/2$, then $|Q| \ge 2n - i - j + 1$.

Proof. Consider a shortest path Q from l_i to r_j. The path Q must cross at least the i lines starting above l_i and the j lines ending above r_j. Crossing any of these lines yields one new face. Hence, the path is at least of length $(i+j+1)$. For $i, j \ge n/2$, consider the lines starting and ending below l_i and r_j. □

Summing the length of the paths in $\mathcal{P}$, we immediately obtain that P has length at least

$$\sum_{i=1}^{\lfloor n/2 \rfloor} |P_i| \ge 2 \sum_{i=1}^{\lfloor n/4 \rfloor} 4i \ge \frac{1}{4} n^2 - n.$$

This ends the proof of the theorem from [1].

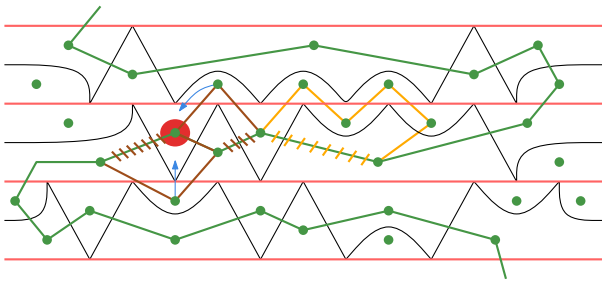


Figure 2: The walls of the tunnels are drawn as straight horizontal lines. Typical rerouting steps are illustrated in orange and brown.

In the following, we give the ideas of how to strengthen this result and obtain a tight constant, i.e., of how to prove Theorem 1. The idea is to prolong the current path P by incorporating sufficiently many faces which are not yet used by the path. We call these unused faces *bad*. The set of bad faces has strong structural properties. Most importantly, the set of bad faces induces a set of paths. These paths can be incorporated until only isolated bad faces remain, see the orange rerouting in Figure 2. This has to be done carefully, due to two reasons: Firstly, some unbounded faces cannot be incorporated. Secondly, after one rerouting, the structure of our path changes and a second rerouting may not be possible.

We use a charging scheme to count the bad faces after the first type of rerouting steps. After eliminating adjacent unused faces, every ramaining unused bad face obtains two units of charge. The charged faces distribute the charge to traversed faces as depicted in Figure 3. It is possible that some traversed faces obtain two units of charge. In these cases, we reroute again or redistribute the charge. A typical rerouting step is illustrated in brown around the red highlighted vertex in Figure 2.

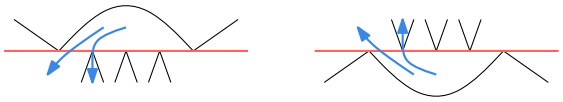


Figure 3: Unused faces give two units of charge to faces in the adjacent tunnel through their leftmost wall-edge and leftmost wall-vertex.

In this way, we can show that the final path P^* has a ratio of traversed and not traversed faces of 2 to 1; implying the desired result.

3 Balanced Bicolored Upper Bound Example

We give a bicolored arrangement $\mathcal{A}$ with $3k$ red and $2k$ blue lines (for odd k) with a longest alternating path of length $14k$. An example for $k = 3$ is given in Figure 4.

Take a regular $3k$-gon (k odd) and construct the red lines as tangents to its edges. As helping lines, construct one line through each vertex and the center of the polygon. Those lines form, up to projective transformation, the Böröczky-example, which minimizes the number of *ordinary crossings* [2, 7]. The helping lines separate the plane into $3k$ double wedges. For each third double wedge draw two blue lines that leave the double wedge only inside the polygon, such that all blue intersections lie within the polygon.

Observation 1 *Consider two adjacent wedges that do not contain a blue line. All faces contained in such a pair of wedges are red, and thus the alternating path cannot* switch *from one wedge to another.*

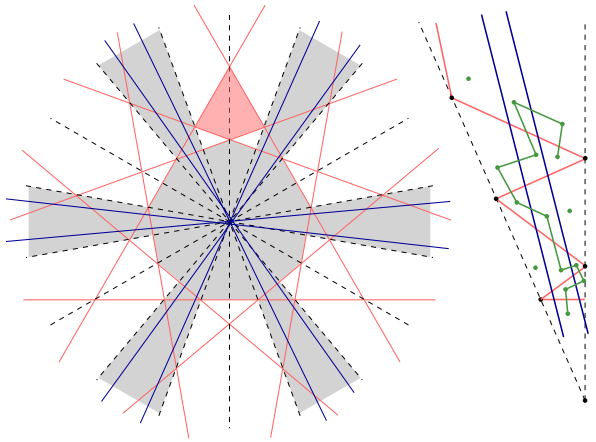


Figure 4: left: Construction for $k = 3$. Each third wedge and the polygon is marked gray. Dotted lines are the helping lines. right: A zoom into one wedge, an alternating path is depicted.

Observation 2 *Consider a wedge that contains blue lines as shown in Figure 4. An alternating path from the middle entering this wedge cannot return to the middle.*

We have to show that the length of a longest path P in the arrangement is bounded by $14k$. The number of bicolored faces adjacent to the boundary of the regular polygon is bounded by $8k$: Each of the $2k$ blue lines crosses the boundary twice and each crossing can be made responsible for two bicolored faces. Consider the faces F traversed by P that are not adjacent to the regular polygon. The faces F lie in at most two wedges by Observation 1 and 2 and are at most $2 \cdot 3k$.

The question if every balanced bicolored line arrangement with k red and k blue lines has an alternating dual path of length $\Omega(n^2)$ remains open.

4 Random Coloring

We will show that a random bicoloring of an arrangement admits a long alternating path, with high probability. Thus each arrangement has a coloring that admits a long alternating path. The general idea follows the uncolored case, but we use tunnels with larger width and need to be more careful, that we can indeed glue paths of each tunnel together.

Let $\mathcal{A}$ be an arrangement of n lines and consider a random coloring of $\mathcal{A}$, i.e., each line is colored red with probability $1/2$ and blue otherwise.

We define the tunnel of width w as

$$\mathcal{T}_i = \bigcup_{j=iw+1}^{(i+1)w} L_j.$$

For simplicity, we assume that w divides n. We denote by $l = n/w - 1$ the index of the last tunnel. Further, we define the *depth* of a face as the minimum number of lines separating a face from an unbounded one. The *outer tunnel* $\mathcal{O}$ is the set of faces of depth at most w. We will suppress the width w in the notation, it will be chosen later appropriately. The idea is similar to the uncolored case: We find a path in each tunnel from left to right or from right to left. The outer tunnel $\mathcal{O}$ is needed to glue those paths together to one.

For technical reasons, we define the following orientation of the dual graph of the arrangement as in [1]. As the dual graph is bipartite, we fix a coloring of the faces with black and white, such that no two adjacent faces obtain the same color. We direct an edge from white to black, if the separating line is blue and we direct an edge from black to white if the separating line is red. Any directed path in this directed graph corresponds to an alternating path in the undirected graph. Thus, in the remainder, we always consider this directed graph. The set of faces that can be reached from a face z in this directed graph is denoted by $\text{reach}(z)$.

Lemma 5 ([1]) *Let E be the set of edges of the arrangement that forms the boundary of $\text{reach}(z)$. Then, the connected components of E are monochromatic.*

We choose a long path in the following way: There exists directed paths L and R in $\mathcal{O}$ from some face in $\mathcal{T}_0$ to $\mathcal{T}_l$ as in Figure 5.

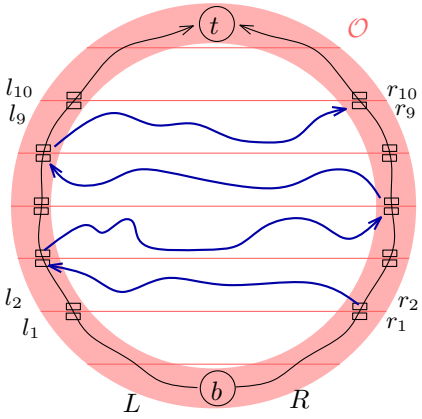


Figure 5: Construction of the path by the tunnel paths.

The path L crosses the wall of $\mathcal{T}_1, \mathcal{T}_2, \ldots$ and so on. We assume, for simplicity, that each wall will be crossed by L exactly once. There are some faces F traversed immediately before and immediately after L crosses a wall. Denote these faces by $\tilde{l}_1, \tilde{l}_2, \ldots$ in the order they are traversed by L. Similarly, we define $\tilde{r}_1, \tilde{r}_2, \ldots$ for the path R. Note that each $\tilde{l}_i, \tilde{r}_i$ is on the top of a tunnel for i odd and on the bottom of a tunnel for i even.

As in the uncolored case, we define paths P_i, if they exist, as follows

$$P_i := \text{a path in tunnel } \mathcal{T}_i \begin{cases} \text{from } \tilde{l}_{2i} \text{ to } \tilde{r}_{2i+1} & i \text{ even}, \\ \text{from } \tilde{r}_{2i} \text{ to } \tilde{l}_{2i+1} & i \text{ odd}. \end{cases}$$

If there is more than one such path pick any. Our long path is $P = P_1 P_2 P_3 \dots$ as depicted in Figure 5.

It remains to show that

(i) the paths L and R exist with high probability,

(ii) the paths P_i exists with high probability, and

(iii) the path P has length $\Omega(n^2/\log n)$.

The proofs of (i) and (ii) are similar. We will show (ii) and sketch (iii).

Lemma 6 *We assume L and R exist as described above. Fix some tunnel $\mathcal{T}_i$ and denote by $\tilde{l}_{2i}$ and $\tilde{l}_{2i+1}$ the first and last face of L in $\mathcal{T}_i$. Similarly let $\tilde{r}_{2i}$ and $\tilde{r}_{2i+1}$ be the bottom and top face of R. Let A_i be the event that P_i does not exist. Then*

$$Pr(A_i) < n^4 \; 2^{-w+3}.$$

Proof. Assume i is even and there exists no path from l_0 to r_1. Then consider the boundary of the reachable region reach(l_0). Since there exists a path from l_0 to l_1, the boundary of reach(l_0) connects the top and bottom wall. Let u and v be the points where the boundary touches the upper and lower wall furthest to the right. By Lemma 5, the points u and v on a connected part of the boundary of the reachable region have the same color, see Figure 6.

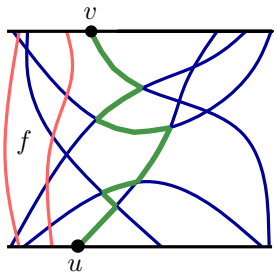


Figure 6: The boundary of *reach*(f) within the tunnel is marked green and is monochromatic.

As u and v are w levels apart, the set $S_{u,v}$ of lines separating u from v contains at least $w-2$ lines. All those lines intersect the boundary between u and v and thus all lines have the same color.

The probability that for two fixed points x, y the set $S_{x,y}$ is monochromatic is upper bounded by $2/(2^{w-2})$.

The complexity of a wall is bounded by $O(n^2)$, thus there are at most Cn^4 pairs of points on walls for some constant C. So the probability that a path within the wall is blocked is bounded by

$$Pr(A_i) \leq \sum_{(x,y)} Pr(S_{x,y} \text{ is mono}) \leq \frac{Cn^4}{2^{w-3}}.$$

□

We choose $w = 6\lceil \log n \rceil + 3$. Using the union bound, we can bound the probability that at least one of the paths P_i does not exist from above by

$$Pr\left(\bigvee A_i\right) \leq \sum Pr(A_i) \quad \leq \sum_{i=1}^{n/w} \frac{Cn^4}{2^{6 \log n}} = o(1).$$

It remains to give a lower bound on the length of P. For this, we extend P_i by adding a shortest path from the start vertex (and the end vertex) to a left (and right) unbounded face within the tunnel $\mathcal{T}_i$. The length of the two extensions can be upper bounded by $2w$ each: at most w faces are needed to reach any unbounded face and another w to ensure an unbounded face within tunnel $\mathcal{T}_i$. By Lemma 4, each path P_i in one of the middle tunnels, i.e., with $l/3 \leq i \leq 2l/3$, contains a linear number of faces:

$$|P_i| \geq 2i(w-1) - 4w \geq 2\frac{n}{3w}\frac{w}{2} - 4w \geq 1/3n - 4w$$

Moreover, the number of these paths is at least $l/3 = \Omega(n/\log n)$. Consequently, P is of length $\Omega(n^2/\log n)$.

Remark: All results hold in the more general setting of *pseudoline arrangements*.

Acknowledgments We thank Nieke Aerts, Stefan Felsner, Heuna Kim and Piotr Micek for interesting and helpful discussions on the topic.

References

[1] O. Aichholzer, J. Cardinal, T. Hackl, F. Hurtado, M. Korman, A. Pilz, R. I. Silveira, R. Uehara, B. Vogtenhuber, and E. Welzl. Cell-paths in mono- and bichromatic line arrangements in the plane. *DMCTS*, 16(3):317–332, 2014.

[2] D.W. Crowe and T.A. McKee. Sylvester's problem on collinear points. *Mathematics Magazine*, pages 30–34, 1968.

[3] T. K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[4] S. Felsner. *Geometric Graphs and Arrangements*. Vieweg, 2004.

[5] Z. Füredi and I. Palásti. Arrangements of lines with a large number of triangles. *Proceedings of the American Mathematical Society*, 92(4):561–566, 1984.

[6] J. E. Goodman and J. O'Rourke. *Handbook of discrete and computational geometry*. CRC press, 2010.

[7] B. Green and T. Tao. On sets defining few ordinary lines. *Discrete & Computational Geometry*, 50(2):409–468, 2013.

[8] H. Tamaki and T. Tokuyama. A characterization of planar graphs by pseudo-line arrangements. In H.W. Leong, H. Imai, and S. Jain, editors, *Algorithms and Computation*, volume 1350 of *LNCS*, pages 133–142. Springer Berlin Heidelberg, 1997.

The Number of Combinatorially Different Convex Hulls of Points in Lines*

Heuna Kim† Wolfgang Mulzer† Eunjin Oh‡

Abstract

Given a sequence $\mathcal{R}$ of planar lines in general position, we can obtain a point set by picking exactly one point from each line in $\mathcal{R}$. We provide exponential upper and lower bounds on the number of combinatorially different convex hulls for point sets that are generated in this manner.

1 Introduction

Recently, geometric problems arising from data imprecision have been studied intensively (e.g., [5]). One popular model is as follows: given a sequence $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ of planar regions, we are promised that our eventual input P contains exactly one point from each R_i. The question is whether the knowledge of $\mathcal{R}$ helps to speed up computations on P. There are many algorithmic results for this setting [1, 2, 4, 6]. Here, we are interested in the *combinatorial* question arising from this model: how much does the knowledge of $\mathcal{R}$ restrict the combinatorial structure of P? We consider the case that (1) $\mathcal{R}$ consists of planar lines, and (2) we measure "variance" by the number of combinatorially different convex hulls. It is known that this setting can be exploited algorithmically [3], and we provide exponential upper and lower bounds on the number of possible convex hulls.

Problem Statement. Let $P = \langle p_1, \ldots, p_n \rangle$ be a sequence of n points in the plane, and let $\mathrm{conv}(P)$ be the convex hull of P. Let $\langle p_{i_1}, p_{i_2}, \ldots, p_{i_k} \rangle$ be the vertices of $\mathrm{conv}(P)$ in clockwise order, such that $i_1 = \min\{i_1, \ldots, i_k\}$. We define the *hull signature* of P as $\sigma(P) = \langle i_1, \ldots, i_k \rangle$.

Consider a sequence $\mathcal{R} = \langle \ell_1, \ldots, \ell_n \rangle$ of n planar lines in *general position*, i.e., every two lines in $\mathcal{R}$ intersect in exactly one point and no three lines in $\mathcal{R}$ have a common intersection. A sequence $P = \langle p_1, \ldots, p_n \rangle$ of points in the plane is *restricted to* $\mathcal{R}$ if $p_i \in \ell_i$, for $i = 1, \ldots, n$. Given $\mathcal{R}$, we would like to study the set $C(\mathcal{R}) = \{\sigma(P) \mid P \text{ is restricted to } \mathcal{R}\}$ of all hull signatures that can be generated by point sequences restricted to $\mathcal{R}$. We also define $c(\mathcal{R}) = |C(\mathcal{R})|$ and $c(n) = \max_{\mathcal{R}} c(\mathcal{R})$, where $\mathcal{R}$ ranges over all sequences of n planar lines in general position. We provide upper and lower bounds on $c(n)$.

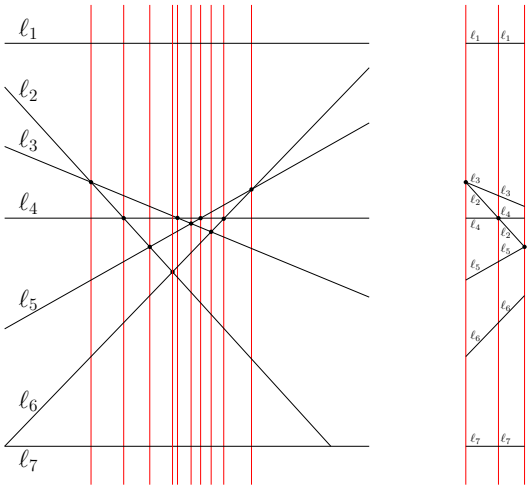


Figure 1: (left) A lower bound example with seven lines. The red lines illustrate the subdivision of the plane into $O(n^2)$ slabs. (right) Between the second and the third slab, ℓ_2 and ℓ_4 change position.

2 Lower Bound

Theorem 1 *For any $n \geq 3$, there exists a sequence $\mathcal{R}$ of n lines in the plane with $c(\mathcal{R}) = \Omega(n^2 3^n)$.*

Proof. Let $\langle \ell_2, \ell_3, \ldots, \ell_{n-1} \rangle$ be a sequence of $n-2$ planar lines in general position, and let $Q = \{\ell_i \cap \ell_j \mid 2 \leq i < j \leq n-1\}$ be their intersection points. Set $m = |Q|$. By general position, we have $m = \binom{n-2}{2}$. We endow the plane with a Cartesian coordinate system such that the points in Q have pairwise distinct x-coordinates. We pick two additional lines ℓ_1 and ℓ_n so that all points in Q lie below ℓ_1 and above ℓ_n along the y-axis, and we set $\mathcal{R} = \langle \ell_1, \ell_2 \ldots, \ell_{n-1}, \ell_n \rangle$; see Fig. 1. We will show that $c(\mathcal{R}) = \Omega(n^2 3^n)$.

For this, we subdivide the plane into *vertical slabs* as follows: let $a_1, \ldots, a_m$ be the sorted sequence of x-coordinates of the points in Q. For $j = 1, \ldots, m-1$, we define the *jth vertical slab* $V_j = \{(x, y) \in \mathbb{R}^2 \mid a_j < x < a_{j+1}\}$. By general position, every vertical slab intersects all lines in $\mathcal{R}$, and by definition, no slab contains an intersection point from Q. Furthermore,

*This work was partially supported by research grant AL 253/8-1 from Deutsche Forschungsgemeinschaft (German Science Association).

†Institut für Informatik, Freie Universität Berlin, Germany. heunak@mi.fu-berlin.de,mulzer@inf.fu-berlin.de

‡Dept. Computer Science and Engineering, Pohang University of Science and Technology jin9082@postech.ac.kr

in every vertical slab the line ℓ_1 lies above all other lines and the line ℓ_n lies below all other lines.

Now, we consider for each slab V_j the point sequences that are restricted to the line segments in V_j. More precisely, define $\mathcal{R}_j = \langle \ell_{1j}, \ldots, \ell_{nj} \rangle$ as the sequence of line segments $\ell_{ij} = \ell_i \cap V_j$, for $i = 1, \ldots n$. We have $C(\mathcal{R}_k) \subseteq C(\mathcal{R})$. Next, we will provide a lower bound on how many distinct hull signatures $C(\mathcal{R}_j)$ contributes to $C(\mathcal{R})$.

We begin with $C(\mathcal{R}_1)$. Fix two arbitrary points $p_1 \in \ell_{11}$ and $p_n \in \ell_{n1}$. The line segment p_1p_n lies inside V_1 and intersects all line segments ℓ_{i1}, for $i = 2, \ldots n-1$. By construction, p_1 and p_n always appear on $\text{conv}(P)$, no matter how we pick $p_2, \ldots, p_{n-1}$ restricted to $\ell_{21}, \ldots, \ell_{(n-1)1}$. For every other point p_i, we have three choices: we can pick $p_i \in \ell_{i1}$ such that (i) the index i does not occur in the hull signature $\sigma(P)$ (i.e., p_i is not on $\text{conv}(P)$); (ii) the index i occurs in $\sigma(P)$ before n; or (iii) the index i occurs in $\sigma(P)$ after n. The choice is not completely free: to avoid degeneracies, $\text{conv}(P)$ must have at least three vertices, so at least one other point needs to appear in $\sigma(P)$. Thus, the number of distinct hull signatures for point sequences restricted to $\mathcal{R}_1$ is at least $3^{n-2} - 1$.

Now fix $j \in \{2, \ldots, m-1\}$ and consider $C(\mathcal{R}_j)$. We give a lower bound on $\left| C(\mathcal{R}_j) \setminus \bigcup_{k=1}^{j-1} C(\mathcal{R}_k) \right|$, the number of hull signatures generated by point sequences restricted to $\mathcal{R}_j$ that are not generated by a point sequence restricted to a previous slab. By construction, the left boundary of $\overline{V_j}$ contains exactly one point $q \in Q$. Let q be the intersection of two lines $\ell_a, \ell_b \in \mathcal{R}$, such that ℓ_a is above ℓ_b to the left of q and ℓ_a is below ℓ_b to the right of q. We pick two arbitrary points $p_1 \in \ell_{1j}$ and $p_n \in \ell_{nj}$. As before, p_1 and p_n always appear on $\text{conv}(P)$, the line segment p_1p_n is contained in V_j, and all other line segments ℓ_{ij} intersect p_1p_n, for $i = 2, \ldots, n-1$. Next, we take points $p_a \in \ell_{aj}$ and $p_b \in \ell_{bj}$ to the left of p_1p_n. As long as p_a and p_b lie on $\text{conv}(P)$, the signature $\sigma(P)$ cannot appear in $\bigcup_{k=1}^{j-1} C(\mathcal{R}_k)$, since in all previous slabs we have that if a and b both occur after n in $\sigma(P)$, then b must come before a. In $C(\mathcal{R}_k)$, however, a comes before b in this case, since ℓ_a and ℓ_b have switched. For all other points p_i, there are again three choices: the index i may appear before or after n in $\sigma(P)$, or it may be absent from $\sigma(P)$. Thus, the slab V_j contributes at least 3^{n-4} new signatures to $\bigcup_{k=1}^{j} C(\mathcal{R}_k)$. There are $\Omega(n^2)$ slabs, so $c(\mathcal{R}) = \Omega(n^2 3^n)$. □

3 Upper Bound

Let $\mathcal{R} = \langle \ell_1, \ldots, \ell_n \rangle$ be a sequence of n planar lines in general position. We denote by $A(\mathcal{R})$ the *arrangement* of $\mathcal{R}$, i.e., the subdivision of the plane into *cells*, *edges* and *vertices* induced by the lines in $\mathcal{R}$: a *cell* is a maximal connected component of $\mathbb{R}^2 \setminus \bigcup_{\ell \in \mathcal{R}} \ell$; an *edge* is a maximal component of a line in $\mathcal{R}$ that does not belong to any other line in $\mathcal{R}$; and a *vertex* is the intersection of two lines in $\mathcal{R}$.

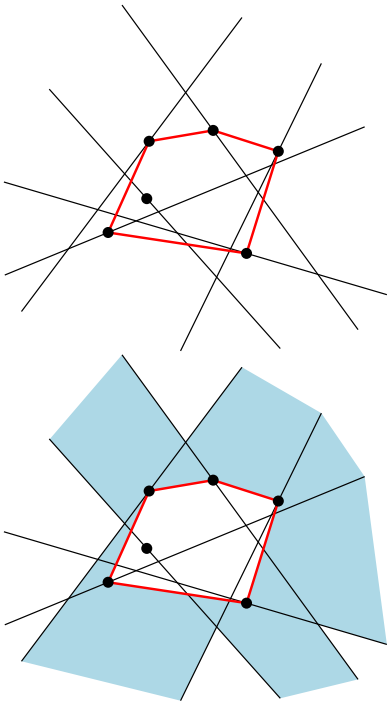


Figure 2: (top) A set $\mathcal{R}$ of lines, a point set P restricted to $\mathcal{R}$, and $\text{conv}(P)$. (bottom) The outer zone of $\text{conv}(P)$. There are 11 outer regions, and the complexity of the outer zone is 39.

Let $P = \langle p_1, \ldots, p_n \rangle$ be a sequence of n points restricted to $\mathcal{R}$. We may assume that no point in P coincides with a vertex of $A(\mathcal{R})$ (otherwise we can perturb P slightly). To distinguish them from edges of $A(\mathcal{R})$, we call the edges of $\text{conv}(P)$ *arcs*. Let $\langle e_1, \ldots, e_k \rangle$ be the arcs of $\text{conv}(P)$, in clockwise order, where e_1 comes after the leftmost point of $\text{conv}(P)$ in clockwise order. For an arc e_i and a cell F of $A(\mathcal{R})$, we say that e_i *properly crosses* F if the relative interior of e_i has nonempty intersection with F and that e_i *touches* F if $e_i \cap \overline{F} = v$, where $v = \overline{e_i} \cap \overline{e_{i+1}}$. The arc e_i *crosses* F if e_i properly crosses F or touches F. Let $\mathcal{F} = \langle F_1, \ldots, F_a \rangle$ be the sequence of cells in $A(\mathcal{R})$ that are crossed by $\langle e_1, \ldots, e_k \rangle$, in clockwise order: first the cells crossed by e_1, then the cells crossed by e_2, etc. The same cell may occur several times in $\mathcal{F}$, but each occurrence is due to one *crossing edge* e_i.

Since the vertices of $\text{conv}(P)$ lie on the edges of $A(\mathcal{R})$, each cell $F_j \in \mathcal{F}$ is divided into at most two parts by the corresponding crossing edge e_i. We define the *outer region* Z_j as the component of $F_j \setminus e_i$ whose interior does not intersect $\text{conv}(P)$. If F_j is touched by e_i, we call Z_j a *touched outer region* and F_j a *touched cell*. In this case, we have $Z_j = F_j$. The sequence $\mathcal{Z} = \langle Z_1, \ldots, Z_a \rangle$ is called the *outer zone* of

conv(P) in $A(\mathcal{R})$. Each outer region $Z \in \mathcal{Z}$ is a (possibly unbounded) convex polygon. For a non-touched outer region Z, exactly one edge of Z belongs to the boundary of conv(P). We call it the *supporting arc* of Z. All other edges of Z are (possibly unbounded) parts of edges of $A(\mathcal{R})$. For a touched outer region Z, no edge is part of the boundary of *conv*(P). Instead, exactly one edge e of Z is intersected by conv(P). We split e into two subedges, each being a maximal connected component of $e \setminus \text{conv}(P)$. The edges of Z consist of the two subedges of e and the other edges of $A(\mathcal{R})$ incident to Z. We call $e \cap \text{conv}(P)$ the *supporting vertex* of Z. A *vertex* of a (touched or non-touched) outer region Z is a vertex of $A(\mathcal{R})$ incident to Z. In particular, the supporting vertex or the endpoints of the supporting arc are not vertices of Z.

The *complexity* of an outer region Z is defined as the number of edges of Z_j other than the supporting arc. The *complexity of the outer zone* $\mathcal{Z}$ is the sum of the complexities of the outer regions, see Fig. 2.

Lemma 2 *The outer zone complexity is at most* $8n$.

Proof. Let ℓ_i be a line in $\mathcal{R}$. By construction, conv(P) intersects ℓ_i, so $\ell_i \setminus \text{conv}(P)$ consists of exactly two unbounded connected components, the two *rays corresponding to* ℓ_i. We orient these rays towards infinity, and we denote by $R = \langle r_1, \ldots, r_{2n} \rangle$ the sequence of all rays that correspond to lines in $\mathcal{R}$.

Consider an outer region Z. Every edge e of Z other than the supporting arc is part of a single ray from R, and we orient e in the same direction as the underlying ray. Now every vertex of Z has either (i) two outgoing incident edges (*out-out* vertex); (ii) two incoming incident edges (*in-in* vertex); or (iii) one incoming and one outgoing incident edge (*in-out* vertex).

Lemma 3 *Let Z be an outer region. Then Z contains no out-out vertex and at most one in-in vertex. If Z has an in-in vertex, then Z is bounded.*

Proof. Suppose that Z contains an out-out vertex v, and let r_1 and r_2 be the two rays with $v = r_1 \cap r_2$. Since Z lies in a cell of $A(\mathcal{R})$, Z is completely contained in the wedge W bounded by the subrays of r_1 and r_2 beginning in v. However, conv(P) is incident to the start vertices of r_1 and r_2, so W does not contain conv(P). It follows that Z cannot be part of the outer zone, a contradiction; see Fig. 3(top).

Next, suppose that Z contains two in-in vertices v_1 and v_2. Suppose further that v_1 comes before v_2 in clockwise order along Z after the supporting arc. All edges between v_1 and v_2 (in clockwise order) are oriented, and both v_1 and v_2 are in-in, so there is at least one out-out vertex between v_1 and v_2; see Fig. 3 (middle). We have just seen that this is impossible.

Finally, suppose that Z contains an in-in vertex v, and let r_1 and r_2 be the two rays with $v = r_1 \cap r_2$.

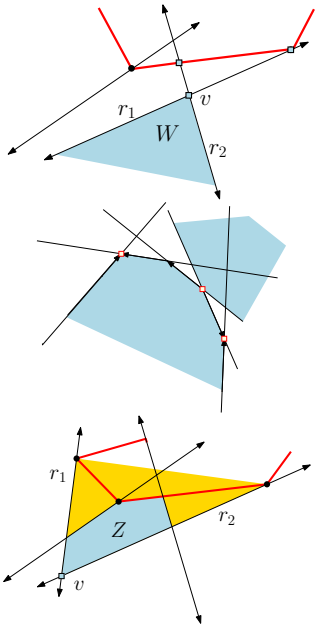


Figure 3: (top) A wedge W bounded by an out-out-vertex v does not contain conv(P). (middle) There cannot be two in-in vertices. (bottom) The outer region Z that contains an in-in vertex v is bounded.

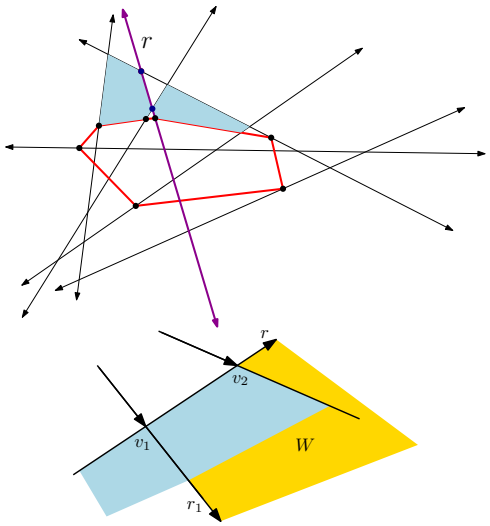


Figure 4: (top) The ray r is charged twice. (bottom) There cannot be two in-out vertices whose outgoing edges lie on the same side of a ray r.

Then Z is completely contained in the region that is bounded by the subrays of r_1 and r_2 from their respective starting points to v, and the boundary of conv(P) between those starting points. It follows that Z is bounded; see Fig. 3(bottom). □

Lemma 4 *There are at most* $4n$ *in-out vertices.*

Proof. Let v be an in-out vertex and let r be the ray that supports the incoming edge of v. We charge v

to r, and we claim that every ray is charged at most twice in this manner; see Fig. 4(top). Indeed, suppose there is a ray r that supports incoming edges for three in-out vertices. Then r contains two in-out vertices v_1 and v_2 such that the outgoing edges for v_1 and v_2 lie on the same side of r. Suppose that v_1 comes before v_2 along r, and let r_1 be the ray supporting the outgoing edge from v_1. Since the rays are directed towards infinity, r and r_1 bound an infinite wedge W with apex v_1. Furthermore, the interior of W is disjoint from $\mathrm{conv}(P)$ and thus does not contain any outer region. This contradicts the assumption that the outgoing edge from v_2 extends into W; see Fig. 4 (bottom). Since there are $2n$ rays, the bound on the number of in-out vertices follows. □

Let Z be an outer region and let n_Z be the number of vertices on Z. If Z is bounded, the complexity of Z is $1 + n_Z$. If Z is unbounded, the complexity of Z is $2 + n_Z$. Hence, the total complexity of the outer zone $\mathcal{Z} = \langle Z_1, \ldots, Z_a \rangle$ is $a + n_{\text{i-o}} + n_{\text{i-i}} + n_{\text{o-o}} + a_u$, where $n_{\text{i-o}}$, $n_{\text{i-i}}$, and $n_{\text{o-o}}$ denotes the number of in-out, in-in, and out-out vertices, and a_u is the number of unbounded outer regions. By Claim 3, $n_{\text{i-i}} + n_{\text{o-o}} + a_u \leq a$, and by Claim 4, $n_{\text{i-o}} \leq 4n$. Thus, the complexity of $\mathcal{Z}$ is at most $2a + 4n \leq 8n$, since $\mathrm{conv}(P)$ intersects every line of $\mathcal{R}$ as most twice, so $a \leq 2n$. □

Theorem 5 *Let $\mathcal{R} = \langle \ell_1, \ldots, \ell_n \rangle$ be sequence of n planar lines in general position. Then $c(\mathcal{R}) = O(n^2 137^n)$.*

Proof. Let P be a sequence of points restricted to $\mathcal{R}$ and let $\mathcal{Z} = \{Z_1, \ldots, Z_a\}$ be the outer zone of $\mathrm{conv}(P)$. To reconstruct the outer zone, it suffices to know (i) the edge e of Z_1 that follows Z_1's supporting arc in clockwise order; and (ii) for $j = 1, \ldots, a$, the complexity z_j of Z_j. Indeed, using this information, we can reconstruct the outer zone and obtain a set $\mathcal{C}$ of candidate locations for the convex hull vertices as follows: $\mathcal{C}$ is initialized as the empty set. Starting from e, we walk for $z_1 - 1$ steps in clockwise direction along the boundary of the corresponding cell in $A(\mathcal{R})$ (when taking a step on an unbounded edge of the cell, we proceed to the other unbounded of of the cell). Then we add the current edge e' to $\mathcal{C}$, if $e' \notin \mathcal{C}$, and cross to the neighboring cell of $A(\mathcal{R})$. Next, we continue for $z_2 - 1$ steps in clockwise direction along the boundary of the current cell. After that, we add the current edge into $\mathcal{C}$ if the edge is not contained in $\mathcal{C}$, and change cells in $A(\mathcal{R})$. We continue until we reach the vertex on e again.

To reconstruct $\mathrm{conv}(P)$ from the candidate set, we need the information about the vertices of $\mathrm{conv}(P)$. Let $\mathcal{C}_\ell = \{e \in \mathcal{C} \mid e \subset \ell\}$, for $\ell = \mathcal{R}$. Since each line $\ell \in \mathcal{R}$ intersects the boundary of $\mathrm{conv}(P)$ at most twice, $|\mathcal{C}_\ell| \leq 2$. Let e_1 and e_2 be elements of $\mathcal{C}_i$. For $\ell \in \mathcal{R}$, an indicator $b_\ell \in \{1, 2, 3\}$ represents whether (1) $\mathrm{conv}(P)$ has a vertex in e_1; (2) $\mathrm{conv}(P)$ has a vertex in e_2; or (3) $\mathrm{conv}(P)$ has no vertex in ℓ.

Thus, for fixed a, the total number of combinatorially different convex hulls can be estimated as

$$2e_{\mathcal{R}} \cdot 3^n \cdot C_{\mathcal{Z}},$$

where $e_{\mathcal{R}}$ denotes the number of edges of $A(\mathcal{R})$, the second term counts the number of indicator vectors (b_ℓ), and $C_{\mathcal{Z}}$ denotes the number of *complexity vectors* $\langle c_1, \ldots, c_a \rangle$. We have $e_{\mathcal{R}} = O(n^2)$. Furthermore, by Lemma 2, the number of complexity vectors is bounded by the number of vectors $(z_1, \ldots, z_a)$ with $z_i \in \{2, \ldots, n\}$ and $\sum_{i=1}^{a} z_i \leq 8n$, which is at most $\binom{8n-a-1}{a-1}$. Thus, for fixed a, the number of combinatorially different convex hulls is at most

$$O\left(n^2 3^n \binom{8n-a}{a}\right).$$

Since this expression grows exponentially with a for $a \in \{1, \ldots, 2n\}$, the total number of combinatorially different convex hulls is asymptotically dominated by the term for $a = 2n$, and it is at most

$$O\left(n^2 3^n \binom{6n}{2n}\right) = O\left(n^2 3^n 2^{\frac{6n}{3}\log 3 + \frac{2\cdot 6n}{3}\log\frac{3}{2}}\right)$$
$$= O\left(n^2 3^n 3^{2n} (3/2)^{4n}\right) = O\left(n^2 137^n\right).$$

□

Acknowledgments. This work was begun at the 17th Korean Workshop on Computational Geometry. We would like to thank the organizers and participants for creating a pleasant and productive research environment. W.Mulzer partially supported by DFG grant MU 3501/1. H.Kim supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

References

[1] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for Delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

[2] O. Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. *J. Comput. Geom. (JoCG)*, 2(1):30–45, 2011.

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

[4] M. van Kreveld, M. Löffler, and J. S. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM Journal on Computing*, 39(7):2990–3000, 2010.

[5] M. Löffler. *Data imprecision in computational geometry.* PhD thesis, 2009.

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

Distinct distances between points and lines

Micha Sharir* Shakhar Smorodinsky† Claudiu Valculescu‡ Frank de Zeeuw‡

Abstract

We show that for m points and n lines in $\mathbb{R}^2$, the number of distinct distances between the points and the lines is $\Omega(m^{1/5}n^{3/5})$, as long as $m^{1/2} < n < m^2$. We also show that for any m non-collinear points, the number of distances between these points and the lines spanned by them is $\Omega(m^{4/3})$.

1 Introduction

Let P be a set of m distinct points and L a set of n distinct lines in the plane. We write $I(P, L)$ for the number of pairs $(p, \ell) \in P \times L$ such that p lies on ℓ. Denote by $I(m, n)$ the maximum value of $I(P, L)$ over all sets with $|P| = m$, $|L| = n$. A classical result of Szemerédi and Trotter [11] is the tight bound $I(m, n) = \Theta\left(m^{2/3}n^{2/3} + m + n\right)$. Two other central questions in combinatorial geometry were introduced by Erdős [4]: *repeated distances* and *distinct distances*. Given m points, the number of pairs of points at some fixed distance is known to be $O(m^{4/3})$ [9], but the best known lower bound is only $\Omega(m^{1+\frac{c}{\log\log m}})$ [4]. The minimum number of distinct distances determined by m points has recently been shown by Guth and Katz [5] to be $\Omega(m/\log m)$, which almost matches the upper bound $O(m/\sqrt{\log m})$.

In this paper we consider questions similar to those above, but for distances *between points and lines*.

In fact, the point-line incidence question can be viewed as a special instance of a repeated distance problem between points and lines. Specifically, the Szemerédi-Trotter result bounds the number of point-line pairs such that the point is at distance 0 from the line. It is an easy exercise to show that the same bound holds for any fixed distance, by replacing each line with the two lines at that distance from it.

Distinct point-line distances. Our first main result concerns distinct distances between m points and n lines in the plane. We write $D(m, n)$ for the minimum number of point-line distances determined by a set of m points and a set of n lines in $\mathbb{R}^2$.

Theorem 1 *For $m^{1/2} < n < m^2$ we have*

$$D(m, n) = \Omega\left(m^{1/5}n^{3/5}\right).$$

In fact, our proof yields a stronger statement: For any set P of m points, and any set L of n lines in the plane, with m and n as above, there always exists a point $p \in P$ such that the number of distinct distances from p to L satisfies the bound in Theorem 1.

This bound is still far from the upper bound $D(m, n) \leq n/2$, which follows from the following construction: Take horizontal lines $y = j$ for $j = 1, \ldots, n$, and place all the points on the median line $y = n/2+1/2$. In contrast with the repeated distances question, the distinct distances variant seems harder for point-line distances than for point-point distances, and the lower bound that we are able to derive is inferior to that of [5]. Nevertheless, we hope that our work will trigger further research into this problem.

Spanned lines. Our second main result is a lower bound on the number of distinct point-line distances between points and their *spanned* lines (the lines passing through at least two of the points). We assume that the points are not collinear, for otherwise there is only one distance, namely 0. This question has a different flavor, because the number of lines spanned by m non-collinear points varies from m to $\binom{m}{2}$. When the points span many lines, Theorem 1 gives a good bound, but when the points span few lines, we have to use a different approach. We write $H(m)$ for the minimum number of distances between m non-collinear points in $\mathbb{R}^2$ and the lines spanned by these points.

Theorem 2 *We have*

$$H(m) = \Omega\left(m^{4/3}\right).$$

The upper bound $H(m) = O(m^2)$, again far from our lower bound, follows from a simple construction: Place $m - 1$ points on a line, and one off the line.

A different way to view this problem is as a question about distinct values of a function of triples of points, i.e., a function of triangles. Specifically, for an ordered triple (a, b, c) of points in $\mathbb{R}^2$, the distance from a to the line spanned by b and c is a *height* of the triangle abc. See [3, Section 6.2] for a discussion of several related problems.

*Tel Aviv University, Israel. Supported by BSF Grant 2012/229, by ISF Grant 892/13, by the I-CORE program (Center No. 4/11), and by the Hermann Minkowski–MINERVA Center for Geometry.

†Ben-Gurion University, Israel. Supported by Grant 1136/12 from the Israel Science Foundation.

‡EPFL, Switzerland. Partially supported by Swiss National Science Foundation Grants 200020-144531 and 200021-137574.

2 Distances between points and lines

In this section we prove Theorem 1. First, we introduce a simplified version of the argument in Section 2.1, and then we give the full proof in Section 2.2.

2.1 A first bound

Let P be a set of m points and L a set of n lines. Let t be the total number of distinct distances between the m points and the n lines. Around each of the m points, draw at most t circles whose radii are the distances occurring from that point, and let C be the set of these circles. We write $T(L, C)$ for the number of *tangencies*, i.e., pairs $(l, c) \in L \times C$ such that ℓ is tangent to the circle c. Note that $T(L, C) = mn$ since every point-line pair gives rise to one tangency.

We now dualize. Concretely, we rotate the plane so that none of the lines is vertical, and then we map a line $y = ax + b$ to the dual point (a, b). Under this map, an algebraic curve is mapped to the set of points that are dual to the non-vertical tangent lines of the curve; these dual points form an algebraic curve, called the *dual curve*. We refer to the original xy-plane as the *primal plane*, and to the ab-plane as the *dual plane*.

Applying this to our setting, the set L of n lines in the primal plane is mapped to a set L^* of n points in the dual plane, and the set C of (at most) mt circles in the primal plane is mapped to a set C^* of mt algebraic curves in the dual plane.

We observe that these curves have *three degrees of freedom*, in the sense that any two curves intersect in a bounded number of points, and for any three points, the number of curves that contain all three points is bounded. The first part follows from the fact that, in the primal plane, any two circles have at most four common tangent lines. The second part corresponds to the fact that any three lines in the primal plane are simultaneously tangent to at most four circles.

Given this property, we can apply the Pach-Sharir incidence theorem [6] to get

$$I(L^*, C^*) = O(n^{3/5}(mt)^{4/5} + n + mt).$$

On the other hand, we have $I(L^*, C^*) = T(L, C) = mn$. Comparing these bounds gives either $mn = O(m^{4/5}n^{3/5}t^{4/5})$, so $t = \Omega(m^{1/4}n^{1/2})$; or $mn = O(n)$, so $m = O(1)$; or $mn = O(mt)$, so $t = \Omega(n)$. Thus

$$D(m, n) = \Omega\left(m^{1/4}n^{1/2}\right),$$

unless $m = O(1)$ or $m = \Omega(n^2)$.

In fact, we could obtain a better bound using the improved incidence bound due to Agarwal et al. [1], although it requires some work to argue that the dual curves satisfy the conditions in that paper. We omit this analysis here, since the refined argument in the next subsection gives an even better bound.

2.2 Proof of Theorem 1

We start, as in Section 2.1, by reducing the problem to counting line-circle tangencies, and then dualize. Instead of directly using an incidence bound for the dual curves, we derive a better bound by taking a closer look at the structure of the problem. Our approach is similar to that used by Székely [10] to prove the bound $\Omega(m^{4/5})$ on the number of distinct point-point distances.

Let P be a set of m distinct points and L a set of n distinct lines in the plane. Again, let t be the number of distinct distances, draw at most t circles around every point, and denote the resulting set of circles by C. As before we have $T(L, C) = mn$.

In the dual plane we have a set L^* of n points. The dual curve c^* of a circle c is the locus of all points (a, b) dual to lines that are tangent to c. If c is centered at a point $p = (p_1, p_2)$ and has radius r, then the equation in a, b that defines c^* is $|p_2 - p_1a - b|/\sqrt{1 + a^2} = r$, or

$$(p_2 - p_1a - b)^2 - r^2(1 + a^2) = 0.$$

This is the equation of a hyperbola. We treat each branch of the hyperbola as a separate curve, and we let C^* be the set of these $2mt$ hyperbola branches.

To bound the number $I(L^*, C^*)$ of incidences between L^* and C^*, we draw a topological (multi-)graph G in the dual plane with vertex set L^*. We assume without loss of generality that each hyperbola branch in C^* contains at least two points of L^*. Indeed, we can discard all curves of C^* containing at most one point of L^*, thereby discarding at most $2mt$ incidences.

For every curve in C^*, we connect each pair of consecutive points of L^* on that curve by an edge drawn along the portion of the curve between the two points. Write E for the set of edges obtained in this way. The number of edges on each curve of C^* is exactly one less than the number of points on it, so overall the number of edges in G satisfies

$$|E| \geq I(L^*, C^*) - 2mt.$$

Note that an edge can have high multiplicity, when many curves of C^* pass through its two endpoints, and the endpoints are consecutive on each of these curves. This situation corresponds to the case in the primal plane where we have many circles touching a pair of lines, and the corresponding tangencies are consecutive on each of the circles.

We define a parameter s, to be chosen later. Let E_1 denote the set of edges with multiplicity at most s and let E_2 denote the set of edges with multiplicity larger than s. In order to bound $|E_1|$ we use the crossing lemma (see [10]), which states that a graph G with n vertices, e edges, and maximum edge multiplicity s, has $\Omega(e^3/sn^2)$ edge crossings in any drawing, unless $e < 4ns$. We apply it to the graph with vertex set L^*

and edge set E_1. Since any two hyperbolas intersect in at most four points, the total number of crossings between curves in C^* is at most $4 \cdot \binom{mt}{2} = O(m^2t^2)$. Combining the two bounds on the number of crossings, we get

$$\frac{|E_1|^3}{sn^2} = O(m^2t^2),$$

so, taking into account the case $|E_1| < 4ns$,

$$|E_1| = O(m^{2/3}n^{2/3}t^{2/3}s^{1/3} + ns). \tag{1}$$

Next, we consider the edges of E_2. If an edge with endpoints ℓ_1^*, ℓ_2^* has multiplicity larger than x, then the lines ℓ_1 and ℓ_2 in the primal plane have x common tangent circles. The centers of these circles lie on the two angular bisectors defined by ℓ_1, ℓ_2. By the pigeonhole principle, there must be $x/2$ incidences between the m points and one of the bisectors of ℓ_1, ℓ_2.

We charge each edge of E_2 to the incidence between the angular bisector and the center of the circle c dual to the curve that the edge lies on. We claim that each such incidence can be charged at most $2t$ times. Indeed, in the primal plane, consider such an incidence between a point p and and an angular bisector ℓ. There are at most t distinct circles with the same center p, and each of these circles can have at most two pairs of tangent lines such that the angular bisector of those lines is ℓ, *and* such that the tangencies are consecutive. (In this argument, we have accounted for the possibility that ℓ might be the angular bisector of many pairs of lines.)

It follows from the Szemerédi-Trotter theorem (see Section 1) that the number of lines containing at least $s/2$ of the m points is $O(m^2/s^3 + m/s)$, as long as $s/2 > 1$. Then applying the Szemerédi-Trotter theorem to these m points and $O(m^2/s^3+m/s)$ lines gives that there are $O(m^2/s^2+m)$ incidences between these points and lines. Thus we have

$$|E_2| = O\left(\frac{m^2t}{s^2} + mt\right). \tag{2}$$

If $m > n^{1/2}$, we can set $s = m^{4/7}t^{1/7}/n^{2/7}$, since then $s/2 > t^{1/7}/2 > 1$ (recall the lower bound on t from Section 2.1). Adding together (1) and (2) gives

$$|E| = O\left(m^{6/7}n^{4/7}t^{5/7} + m^{4/7}n^{5/7}t^{1/7} + mt\right).$$

Thus the same bound holds for $T(L, C)$. Combining this with $T(L, C) = mn$, we get $t = \Omega(m^{1/5}n^{3/5})$ from the first term, $t = \Omega(m^3n^2)$ from the second term, and $t = \Omega(n)$ from the third term. Thus

$$t = \Omega\left(m^{1/5}n^{3/5}\right),$$

if $m < n^2$, and assuming that $m > n^{1/2}$. This completes the proof of Theorem 1. □

Note that in the proof above we could let t be the maximum number of distances from a point to a line. Then we could conclude that there is a single point such that the number of distances from this point satisfies the bound. On the other hand, the proof in Section 3 does not lead to such a conclusion.

3 Distances between points and spanned lines

We now consider the problem of bounding from below the number of distinct distances between a non-collinear point set P and the lines spanned by P. Write ℓ_{bc} for the line spanned by points b and c, write

$$H(P) = |\{d(a, \ell_{bc}) \mid a, b, c \in P\}|$$

for the number of distances between points of P and lines spanned by P, and write $H(m)$ for the minimum value of $H(P)$ over all point sets P of size m.

For point sets with not too many points on a line, a good bound follows from Theorem 1. However, we aim for a reasonably good bound that also holds when many points are collinear. We reduce it to showing that the rational function $f(x, y) = (x - y)^2/(1 + y^2)$ "expands", in the sense that $f(x, y)$ takes $\Omega(m^{4/3})$ distinct values for x, y in any set of m real numbers. If f were a polynomial, this would follow directly from [7]. To extend the bound $\Omega(m^{4/3})$ to the rational function f, we use the same approach as [7], which originated in [8].

Proof of Theorem 2: By a theorem of Beck [2, Theorem 3.1], there is a constant c such that either the points of P span at least cm^2 distinct lines, or at least cm points of P are collinear.

In the first case, Theorem 1 gives the lower bound

$$H(P) = \Omega\left(m^{1/5}(cm^2)^{3/5}\right) = \Omega\left(m^{7/5}\right).$$

Consider the second case, when $k = cm$ of the points are collinear. Since not all the points are collinear, at least one other point $q \in P$ does not belong to this line. By translating, rotating, and scaling, we can assume that $q = (0, 1)$ and that the other points are on the x-axis, and by removing at most half the points we can assume that they are all on the positive x-axis. We denote them by $p_i = (x_i, 0)$ for $i = 1, \ldots, k$, with all x_i positive, and we set $W = \{x_1, \ldots, x_k\}$.

The interesting distances are those from a point p_i to the line ℓ_{p_jq} spanned by p_j and q. We define

$$f(x, y) = \frac{(x - y)^2}{1 + y^2},$$

so that $f(x_i, x_j)$ equals the square of the distance $d(p_i, \ell_{p_jq})$. In order to obtain a lower bound for the number of point-line distances, it suffices to find a lower bound for the number of distinct values of f, i.e., the cardinality of $f(W) = \{f(x, y) \mid x, y \in W\}$.

Following the setup in [8], we define

$$Q = \{(x, y, x', y') \in W^4 \mid f(x, y) = f(x', y')\}.$$

Writing $f^{-1}(a) = \{(x, y) \in W^2 \mid f(x, y) = a\}$ and using the Cauchy-Schwarz inequality, we obtain

$$|Q| = \sum_{a \in f(W)} |f^{-1}(a)|^2 \geq \frac{k^4}{|f(W)|}.$$

We wish to establish an upper bound on $|Q|$.

We define algebraic curves C_{ij} in $\mathbb{R}^2$ by

$$C_{ij} = \{(z, z') \in \mathbb{R}^2 \mid f(z, x_i) = f(z', x_j)\}.$$

Then $(x_k, x_l) \in C_{ij}$ if and only if $(x_k, x_i, x_l, x_j) \in Q$. Thus, denoting by Γ the set of curves C_{ij}, and by S the set of pairs (x_k, x_l), we have $|Q| = |I(S, \Gamma)|$. It is not hard to show that the curves C_{ij} with $i = j$ contribute at most $O(k^2)$ quadruples, which is a negligible number, so in the rest of the proof we assume that $i \neq j$.

The equation $f(z, x_i) = f(z', x_j)$ is equivalent to

$$z' - x_j = \pm A_{ij} \cdot (z - x_i),$$

where we write $A_{ij} = \sqrt{(1 + x_j^2)/(1 + x_i^2)}$. Every curve C_{ij} is thus the union of two lines in the zz'-plane, given by

$$L_{ij}^+ : z' = A_{ij} z + (x_j - A_{ij} x_i),$$
$$L_{ij}^- : z' = -A_{ij} z + (x_j + A_{ij} x_i).$$

Therefore, we need only consider the two families $\Gamma^+ = \{L_{ij}^+ \mid i \neq j\}$ and $\Gamma^- = \{L_{ij}^- \mid i \neq j\}$. We want to bound $I(S, \Gamma^+ \cup \Gamma^-)$, but we need to deal with the possibility that some of these lines coincide.

Since $I(S, \Gamma) = I(S, \Gamma^+) + I(S, \Gamma^-)$, it suffices to consider only coincidences of lines in Γ^+, and coincidences of lines in Γ^-. Without loss of generality, we focus on the former type.

Suppose that

$$L_{ij}^+ : \ z' = A_{ij} z + (x_j - A_{ij} x_i),$$
$$L_{kl}^+ : \ z' = A_{kl} z + (x_l - A_{kl} x_k)$$

define the same line, for some $(x_i, x_j) \neq (x_k, x_l)$. Considering (x_i, x_j) fixed, (x_k, x_l) would have to satisfy

$$\sqrt{\frac{1 + x_l^2}{1 + x_k^2}} = A_{ij} \qquad \text{and} \qquad x_l - x_k \sqrt{\frac{1 + x_l^2}{1 + x_k^2}} = B_{ij},$$

where we write $B_{ij} = x_j - x_i A_{ij}$. The first equation can be rearranged to $x_l^2 - A_{ij}^2 x_k^2 = A_{ij}^2 - 1$, which defines a nondegenerate hyperbola in the $x_k x_l$-plane (we have $A_{ij}^2 \neq 1$ because $i \neq j$). The second equation can be rearranged to the equation

$$2B_{ij} x_k^2 x_l = (B_{ij}^2 - 1) x_k^2 + x_l^2 - 2B_{ij} x_l + B_{ij}^2,$$

which defines a cubic curve. It is not hard to show that this hyperbola and cubic do not have a common factor, for instance by showing that both are irreducible. Bézout's theorem then implies that they have at most six common points. In other words, a line in Γ^+ or Γ^- occurs with multiplicity at most six.

We thus have an incidence problem for points and lines, with k^2 points and $O(k^2)$ distinct lines, each with multiplicity at most six. The Szemerédi-Trotter theorem (see Section 1) gives

$$\begin{aligned} |I(S, \Gamma^+ \cup \Gamma^-)| &= O((k^2)^{2/3} (k^2)^{2/3} + k^2 + k^2) \\ &= O(k^{8/3}). \end{aligned}$$

Taking into account the discarded quadruples, we have $|Q| = |I(S, \Gamma^+ \cup \Gamma^-)| + O(k^2) = O(k^{8/3})$, so

$$|f(W)| \geq \frac{k^4}{|Q|} = \Omega(k^{4/3}) = \Omega(m^{4/3}),$$

completing the proof of Theorem 2. □

References

[1] P. K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, S. Smorodinsky, Lenses in arrangements of pseudo-circles and their applications, *J. ACM* **51** (2004), 139–186.

[2] J. Beck, On the lattice property of the plane and some problems of Dirac, Motzkin, and Erdős in combinatorial geometry, *Combinatorica* **3** (1983), 281–297.

[3] P. Brass, W. Moser, J. Pach, *Research Problems in Discrete Geometry*, Springer-Verlag, New York, 2005.

[4] P. Erdős, On sets of distances of n points, *Amer. Math. Monthly* **53** (1946), 248–250.

[5] L. Guth, N.H. Katz, On the Erdős distinct distances problem in the plane, *Annals Math.* **181** (2015), 155–190.

[6] J. Pach, M. Sharir, On the number of incidences between points and curves, *Combin. Probab. Comput.* **7** (1998), 121–127.

[7] O.E. Raz, M. Sharir, J. Solymosi, Polynomials vanishing on grids: The Elekes-Rónyai problem revisited, *Amer. J. Math.*, to appear. Also in *Proc. 30th Annu. ACM Sympos. Comput. Geom.* (2014), 251–260.

[8] M. Sharir, A. Sheffer, J. Solymosi, Distinct distances on two lines, *J. Combinat. Theory, Ser. A* **120** (2013), 1732–1736.

[9] J. Spencer, E. Szemerédi, W.T. Trotter, Unit distances in the Euclidean plane, in: *Graph Theory and Combinatorics* (B. Bollobás, ed.), Academic Press, 1984, 293–303.

[10] L. Székely, Crossing numbers and hard Erdős problems in discrete geometry, *Combin. Probab. Comput.* **6** (1997), 353–358.

[11] E. Szemerédi, W.T. Trotter, Extremal problems in discrete geometry, *Combinatorica* **3** (1983), 381–392.

Ramsey numbers for empty convex polygons

Crevel Bautista-Santiago Javier Cano* Ruy Fabila-Monroy† Carlos Hidalgo Toscano‡
Clemens Huemer§ Jesús Leaños¶ Toshinori Sakai‖ Jorge Urrutia**

Abstract

We study a geometric Ramsey type problem where the vertices of the complete graph K_n are placed on a set S of n points in general position in the plane, and edges are drawn as straight-line segments. We define the empty convex polygon Ramsey number $R_{EC}(k,k)$ as the smallest number n such that for every set S of n points and for every two-coloring of the edges of K_n drawn on S, at least one color class contains an empty convex k-gon. A polygon is empty if it contains no points from S in its interior. We prove $17 \leq R_{EC}(3,3) \leq 463$ and $57 \leq R_{EC}(4,4)$. Further, there are three-colorings of the edges of K_n (drawn on a set S) without empty monochromatic triangles. A related Ramsey number for islands in point sets is also studied.

1 Introduction

Ramsey's theorem ensures that for every two-coloring of the edges of the complete graph K_n on a large enough number n of vertices, at least one of the two color classes contains a clique of a given size. The Ramsey number $R(s,t)$ is the smallest number n such that every two-coloring of the edges of K_n contains a clique on s vertices from the first color class or a clique on t vertices from the other color class. Geometric variants of Ramsey's theorem have been studied, see e.g. [9]. When the vertices of K_n are drawn on a set of n points in the plane, and edges as straight-line segments, geometry comes into play by considering crossings of edges. Throughout, we only consider point sets S in general position, meaning sets without three collinear points. For example, in [11] it was shown that for every set S of n points and for every two-coloring of the edges of K_n drawn on S, one color class has non-crossing cycles of lengths $3, 4, \ldots, \left\lfloor \sqrt{n/2} \right\rfloor$. In this work we consider another geometric constraint, namely emptyness. A simple polygon is *empty* if it has no points of S in its interior. The number of empty convex polygons in K_n drawn on sets S of n points have been estimated, see e.g. [1, 2, 7, 10]. We define the empty convex polygon Ramsey number $R_{EC}(s,t)$ as the smallest number n such that for every set S of n points and for every two-coloring of the edges of K_n drawn on S, the first color class contains an empty convex s-gon or the second color class contains an empty convex t-gon. For the case of empty triangles, the bounds $17 \leq R_{EC}(3,3) \leq 463$ are shown. We also prove that there are three-colorings of the edges of K_n, drawn on some point set S, without empty monochromatic triangles; in other words $R_{EC}(3,3,3) = 0$. For the case of empty convex quadrilaterals we can show the lower bound $R_{EC}(4,4) \geq 57$. We were not able to prove an upper bound. Finally we consider a Ramsey number for islands in point sets. An island of a point set S is a subset I of S such that $Conv(I) \cap S = I$. Islands in point sets were also studied in [3, 4, 6]. In our context, an island is a clique formed by a subset of vertices of K_n drawn on S which contains no further point of S in its interior. We remark that the Ramsey number $R(s,t)$ equals the smallest number n such that every two-coloring of the edges of K_n drawn on a set of n points in convex position contains an island on s points in one color class or an island on t points in the other color class. This is, because there, all islands are in convex position. In [13] it was shown that for every set S of n points, the edges of K_n, drawn on S, can be two-colored such that there is no monochromatic island on four points with triangular convex hull. We prove that there are point sets S and a two-coloring of the edges of K_n, drawn on S, such that there is no monochromatic island on four points (regardless of the form of the convex hull). That is, the island Ramsey number for four points $R_I(4,4)$ is zero.

*Universidad Nacional Autónoma de México.

†Departamento de Matemáticas, Cinvestav, D.F. México. Partially supported by Conacyt of Mexico, grant 153984.

‡Departamento de Matemáticas, Cinvestav, D.F. México. Partially supported by Conacyt of Mexico, grant 153984.

§Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, BarcelonaTech, Spain. Research supported by projects Gen. Cat. DGR 2014SGR46 and MINECO MTM2012-30951.

¶Unidad Académica de Matemáticas, UAZ, México. Partially supported by Conacyt of Mexico, grant 179867.

‖Department of Mathematics, School of Science, Tokai University, Japan. Research supported by JSPS KAKENHI Grant Number 24540144.

**Instituto de Matemáticas, Universidad Nacional Autónoma de México.

2 The empty triangle Ramsey number

Theorem 1 *The empty triangle Ramsey number satisfies* $17 \leq R_{EC}(3,3) \leq 463$.

Proof. For the upper bound, we use the fact that every sufficiently large point set in general position contains an empty convex hexagon [8, 14]. Koshelev obtained the current best bound, 463, on the number of points needed to guarantee such an empty convex hexagon [12]. Consider only the complete graph on six vertices K_6 formed by the vertices of this hexagon. Ramsey's theorem tells us that every two-coloring of K_6 contains a monochromatic triangle. Since the hexagon is empty, the monochromatic triangle is so as well. For the lower bound, a two-colored complete geometric graph on 16 vertices without an empty monochromatic triangle is shown in Figure 1.

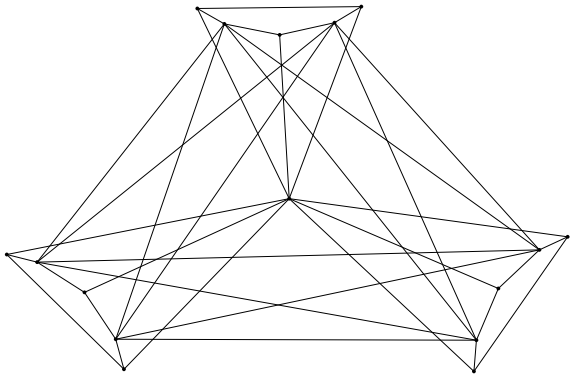


Figure 1: A two-coloring of the edges of K_{16} without an empty monochromatic triangle. Only the edges of one color class are drawn.

□

Theorem 2 *The empty triangle Ramsey number for three-colored complete graphs* $R_{EC}(3,3,3)$ *is zero.*

Proof. We have to present a three-coloring of the edges of the complete geometric graph K_n drawn on a set S of n points. The point set S is the so-called *Horton set* $H(n)$, see e.g. [1, 2, 5, 10], defined recursively as follows: $H(1) = \{(1,1)\}$ and $H(2) = \{(1,1),(2,2)\}$. When $H(n)$ is defined, set

$$H(2n) = \{(2x-1, y) \mid (x,y) \in H(n)\}$$

$$\cup \{(2x, y+3^n) \mid (x,y) \in H(n)\}.$$

In this construction $H(2n)$ is obtained by taking $H(n)$ and a copy of $H(n)$ which is slightly shifted to the right and placed far above the other set $H(n)$. To define an edge-coloring of the complete graph drawn on $H(n)$ we use an auxiliary three-coloring of the vertices of $H(n)$: vertex (x,y) gets color $x \mod 3$. This three-coloring for $H(8)$ is shown in Figure 2. In [5], Theorem 3.3, it was proved that this coloring admits no empty triangles with its three vertices from the same color class. The three-coloring for the edges of K_n is now defined as follows: an edge connecting points (x_1,y_1) and (x_2,y_2) gets color $x_1 + x_2 \mod 3$. Then, a triangle formed by points (x_1,y_1), (x_2,y_2) and (x_3,y_3) is monochromatic if and only if x_1, x_2 and x_3 belong to the same congruence class modulo three. Thus, the vertices of a monochromatic triangle have the same color and from [5] we know that these triangles are not empty. □

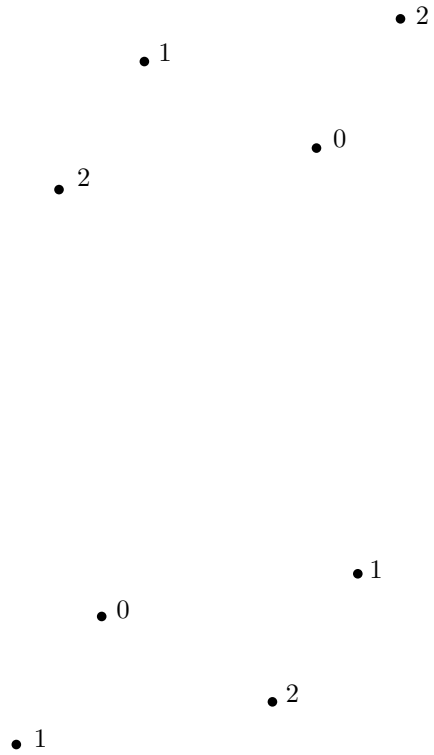


Figure 2: A three-coloring of the vertices of the Horton set $H(8)$.

3 The empty convex quadrilateral Ramsey number

Theorem 3 *The empty convex quadrilateral Ramsey number satisfies* $57 \leq R_{EC}(4,4)$.

Proof. Figure 3 shows a two-coloring of the edges of K_{11} in convex position without an empty convex monochromatic quadrilateral. A drawing of K_{56} (indicated in Figure 4) and a two-coloring of its edges without an empty convex monochromatic quadrilateral is obtained by placing five groups of 11 points (with two-coloring as in Figure 3) in such a way that the 55 points lie on five small semi-circles with centers the vertices of a regular pentagon. Then the last point is placed in the center of this pentagon and connected to the 55 points with the same color as the drawn

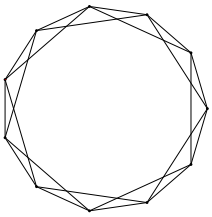


Figure 3: A two-coloring of the edges of K_{11} without an empty convex monochromatic quadrilateral. Only the edges of one color class are drawn.

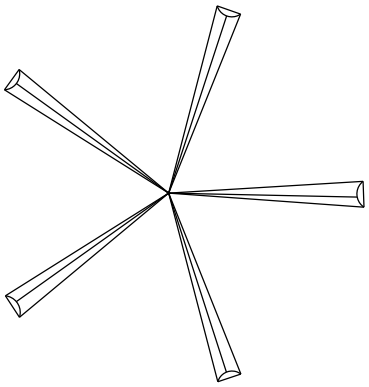


Figure 4: Schematic drawing of K_{56} without an empty convex monochromatic quadrilateral. Only the edges of one color class are indicated.

edges in Figure 3.

□

4 The Ramsey number for islands

Theorem 4 *The island Ramsey number $R_I(4,4)$ is zero.*

Proof. We present a two-coloring of the edges of K_n drawn on the Horton set $H(n)$ without an empty monochromatic K_4. As in the proof of Theorem 2, we start with the auxiliary three-coloring of the vertices of $H(n)$ where vertex (x, y) gets color $x \mod 3$. Now we define a two-coloring for the edges of K_n as follows: an edge connecting points (x_1, y_1) and (x_2, y_2) gets color 0 if $x_1 - x_2 \mod 3 = 0$ and gets color 1 otherwise. In other words, an edge gets color 0 if and only if its two vertices have the same color in the auxiliary vertex coloring. Then, a complete subgraph K_4 is monochromatic if and only if its four vertices have the same color in the auxiliary vertex coloring. Thus, if a K_4 is monochromatic, then from [5] Theorem 3.3, we know that none of its triangles is empty, which implies that this K_4 is not an island.

□

5 Concluding Remarks

An obvious problem left open is to close the gap between lower and upper bound for $R_{EC}(3,3)$. Very interesting would be to prove an upper bound on the empty convex quadrilateral Ramsey number. Computer experiments suggest that it is finite and probably not too large.

References

[1] I. Bárány, Z. Füredi. *Empty simplices in Euclidean space.* Canad. Math. Bull., 30 (1987), 436–445.

[2] I. Bárány, P. Valtr. *Planar point sets with a small number of empty convex polygons.* Stud. Sci. Math. Hung., 41 (2004), 243–269.

[3] C. Bautista-Santiago, J. Cano, R. Fabila Monroy, D. Flores-Peñaloza, H. González-Aguilar, D. Lara, E. Sarmiento, J. Urrutia. *On the connectedness and diameter of a geometric Johnson Graph.* Discrete Mathematics & Theoretical Computer Science 15 (2013), 21–30.

[4] C. Bautista-Santiago, J.M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, I. Ventura. *Computing Optimal Islands.* Operations Research Letters, 39 (2011), 246–251.

[5] O. Devillers, F. Hurtado, G. Kàrolyi, C. Seara. *Chromatic variants of the Erdős-Szekeres Theorem.* Comput. Geom., 26 (2003) 193–208.

[6] R. Fabila-Monroy, C. Huemer. *Covering islands in plane point sets.* Lecture Notes in Computer Science, 7579 (2012), 220–225.

[7] A. García. *A note on the number of empty triangles.* Lecture Notes in Computer Science, 7579 (2012), 249–257.

[8] T. Gerken. *Empty convex hexagons in planar point sets.* Discrete and Computational Geometry 39 (2008), 239-272.

[9] H. Harborth, H. Lefmann. *Coloring arcs of convex sets.* Discrete Mathematics, 220 (2000), 107–117.

[10] J.D. Horton. *Sets with no empty convex 7-gons.* Canad. Math. Bull., 26 (1983), 482–484.

[11] G. Kàrolyi, J. Pach, G. Tóth, P. Valtr. *Ramsey-Type results for geometric graphs II.* Discrete and Computational Geometry, 20 (1998), 375-388.

[12] V.A. Koshelev. *On Erdős-Szekeres problem for empty hexagons in the plane.* Model. Anal. Inform. Sist., 16 (2009), 22-74.

[13] J. Nešetřil, J. Solymosi, P. Valtr. *Induced monochromatic subconfigurations.* In: Contemporary Trends in Discrete Mathematics, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 49, AMS, (1999), 219–227.

[14] C.M. Nicolás. *The empty hexagon theorem.* Discrete and Computational Geometry, 38 (2007), 389–397.

Efficient Spanner Construction for Directed Transmission Graphs*

Haim Kaplan† Wolfgang Mulzer‡ Liam Roditty§ Paul Seiferth‡

Abstract

Let $P \subset \mathbb{R}^2$ be a set of n points, each with an associated radius $r_p > 0$. The *transmission graph* G for P has vertex set P and a directed edge from p to q if and only if q lies in the ball with radius r_p around p. Let $t > 1$. A *t-spanner* H for G is a sparse subgraph such that for any two vertices p and q connected by a path of length l in G, there is a path of length at most $t \cdot l$ from p to q in H. Given G implicitly as points with radii, we show how to compute a t-spanner for G in time $O(n(\log n + \log \Psi))$, where Ψ is the ratio of the largest and smallest radius in P.

1 Introduction

A common model for wireless sensor networks is the *unit-disk graph*: each sensor is modeled by a unit disk, and there is an edge between two sensors iff their disks intersect. Intersection graphs of disks with arbitrary radii have been used to model different transmission radii. These graphs are undirected, while for some networks a directed model would be more appropriate. This motivated Peleg and Roditty [5] to define *transmission graphs*. The vertex set of a transmission graph G is a point set $P \subset \mathbb{R}^2$, where each $p \in P$ has a radius $r_p > 0$. There is a directed edge $\overrightarrow{pq}$ from p to q iff q lies in the disk $D(p)$ of radius r_p around p.

Although transmission graphs are represented succinctly, they may have $\Theta(n^2)$ edges. Thus we would like to approximate them by sparse *spanners*. For $t > 1$, a subgraph $H \subseteq G$ is a *t-spanner* for G if the distance between any two vertices p and q in H is at most t times the distance between p and q in G (cf, e.g., [4]). Fürer and Kasivisawnathan showed how to compute spanners for unit and general disk graphs by adapting the Yao graph [3, 6]. Peleg and Roditty [5] gave a spanner-construction for transmission graphs in any metric of bounded doubling dimension. Except for the unit-disk case, the running times depend on the number of edges. We avoid this dependency and give an efficient algorithm to construct t-spanners for transmission graphs for the planar Euclidean case.

Preliminaries and Results. Let $P \subset \mathbb{R}^2$ be a point set with radii, and let G be its transmission graph. Let $\Phi = \max_{p,q \in P} |pq| / \min_{p \neq q \in P} |pq|$ be the *spread* of P. In §2, we give a construction depending on Φ:

Theorem 1 *Let G be the transmission graph for an n-point set $P \subset \mathbb{R}^2$ with spread Φ. For any $t > 1$, we can find a t-spanner for G in time $O(n(\log n + \log \Phi))$.*

The *radius ratio* $\Psi = \max_{p,q \in P} r_p / r_q$ of P is the ratio of the largest and the smallest radius in P. In §3 we extend our construction to depend on Ψ instead of Φ.

Theorem 2 *Let G be the transmission graph for an n-point set $P \subset \mathbb{R}^2$ with radius ratio Ψ. For $t > 1$, we can find a t-spanner for G in time $O(n(\log n + \log \Psi))$.*

We may assume that $\Psi \leq \Phi$: a radius less than the smallest distance c in P can be set to $c/2$, and a radius larger than the diameter d of P can be set to d.

Our construction uses planar grids. For $i = 0, 1, \ldots$, the *grid at level i*, $\mathcal{Q}_i$, consists of axis-parallel squares of diameter 2^i that partition the plane in grid-like fashion (the *cells*). $\mathcal{Q}_i$ is aligned so that the origin is a grid vertex. The *distance* between two cells is the smallest distance of any two points contained in them. We assume that our computational model can find the grid cell containing a given point in $O(1)$ time.

2 Efficient Spanner Construction

Let $P \subset \mathbb{R}^2$ be a point set with radii, and let Φ be the spread of P. Let G be the transmission graph of P. Our spanner construction is a modification of the Yao graph [6] that takes the disks into account. Ideally, our spanner H should look as follows: we pick a suitable $k \in \mathbb{N}$, and we let $\mathcal{C}$ be a set of k cones with opening angle $2\pi/k$ and the origin as apex that partition the plane. For $q \in P$ and $C \in \mathcal{C}$, let C_q be the translated copy of C with apex q. We pick the closest vertex $p \in P$ in C_q with $q \in D(p)$, and we add the edge $\overrightarrow{pq}$ to H. This gives $O(kn)$ edges, and one can show that H is a t-spanner for $t = 1 + \Theta(1/k)$. This is folklore in the spanners community [2, 5].

Since we do not know how to find these edges quickly, we present an approximate construction with similar properties. We partition each cone C_q into "intervals" obtained by intersecting C_q with annuli centered at q whose inner and outer radii grow exponentially; see Fig. 1. Then we cover each interval with $O(1)$ grid cells whose diameter is "small" compared to the distance between the interval and q.

*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

This gives two properties that help us find an approximately shortest incoming edge for q in C_q: once we have an incoming edge, we need not consider larger intervals, and if there are multiple edges from the same cell, it suffices to pick one of them. Below, we define a

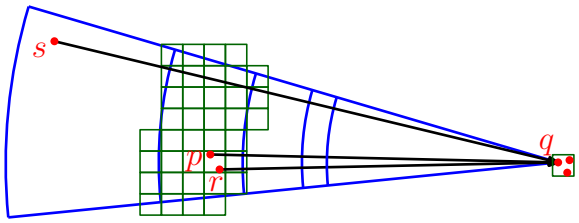


Figure 1: A cone C_q covered by discretized intervals. We only need one of the edges, $\overrightarrow{pq}$ or $\overrightarrow{rq}$, for H.

decomposition of P that represents the discretized intervals by a neighborhood relation between grid cells.

We first give the properties of this decomposition and use it to find the edges for H. Then we prove that H is a t-spanner for an appropriate choice of parameters. Finally, we show how to use a quadtree to find this decomposition and how to implement the main steps in the desired running time.

Let $c > 2$ be a large constant. For a grid cell σ, let m_σ be the point in $P \cap \sigma$ with the largest radius.

Definition 1 *Let G be a transmission graph with vertex set $P \subset \mathbb{R}^2$. A c-separated annulus decomposition for G consists of a finite set $\mathcal{Q} \subset \bigcup_{i=0}^{\infty} \mathcal{Q}_i$ of grid cells, a symmetric* neighborhood relation $N \subseteq \mathcal{Q} \times \mathcal{Q}$, *and assigned sets R_σ for each $\sigma \in \mathcal{Q}$ so that (i) for all $(\sigma, \sigma') \in N$, $\operatorname{diam}(\sigma) = \operatorname{diam}(\sigma')$ and $d(\sigma, \sigma') \in [(c-2)\operatorname{diam}(\sigma), 2c\operatorname{diam}(\sigma))$; and (ii) for every edge $\overrightarrow{pq}$ of G, there is $(\sigma, \sigma') \in N$ with $p \in \sigma$, $q \in \sigma'$, and with either $p \in R_\sigma$ or $q \in D(m_\sigma)$.*

For $\sigma \in \mathcal{Q}$, we define $N(\sigma) = \{\sigma' \mid (\sigma, \sigma') \in N\}$. Property (i) in Def. 1 implies $|N(\sigma)| = O(1)$.

Getting a Spanner. Let $t > 1$ be the desired stretch factor. Depending on t, we choose constants c (separation) and k (number of cones) in a way to be described later. Let $\mathcal{Q}$ be a c-separated annulus decomposition for G. To get a t-spanner $H \subseteq G$, we go through all cones $C \in \mathcal{C}$ and pick all incoming edges for C as in Alg. 1. Instead of searching incoming edges for each point $q \in P$ separately, we group points using the cells of $\mathcal{Q}$. This gives the speed-up required for the desired running time, as shown later. We consider the cells $\sigma \in \mathcal{Q}$ by increasing diameter, and we search incoming edges for points in $\sigma \cap P$ without incoming edges so far. These are the *active* points. Initially all points are active. Fix a pair $(\sigma, \sigma') \in N$ and let Q and R as in Alg. 1. We find for each point in Q one incoming edge whose other endpoint is in R, if such an edge exists (edge selection). Having Q sorted (line 5) allows us to find these edges efficiently (see Lemma 8).

```
1 Set all points in P to active
2 foreach σ ∈ Q by increasing diameter do
3     foreach σ' ∈ N(σ) do
4         Q ← all active q ∈ σ ∩ P with C_q ∩ σ' ≠ ∅
5         Sort Q in x/y-direction // preproccess
6         R ← R_σ' ∪ {m_σ'}
7         // edge selection
8         For all q ∈ Q find r ∈ R with q ∈ D(r), if
          it exists, and add r→q to H
9     Make all q ∈ Q with incoming edges inactive
```

Algorithm 1: Finding edges for P in a cone $C \in \mathcal{C}$.

For each $C \in \mathcal{C}$ and $q \in P$, at most one cell $\sigma \in \mathcal{Q}$ with $q \in \sigma$ gives incoming edges for q: q becomes inactive after processing σ. Since $|\mathcal{C}| = k$ and $|N(\sigma)| = O(1)$, q has $O(k)$ incoming edges, and H has $O(n)$ edges. Next, we show that H is a t-spanner. For this, we need three technical lemmas: Lemma 3 deals with the imprecision due to the grid. Let $\overrightarrow{pq}$ be an edge of G contained in the cone C_q. We prove that if we slightly increase the opening angle of C_q, Alg. 1 picks at least one edge $\overrightarrow{rq}$ contained in the larger cone. Lemmas 4 and 5 let us bound the distance between the endpoints r and p. Lemma 5 is due to Bose et al [1]. For space reasons, we omit the proofs of Lemmas 3 and 4.

Lemma 3 *Let $k \geq 8$ and $c > 3 + 2/(\sin \pi/k)$. Given $i \in \mathbb{N}_0$ and cells $\sigma, \sigma' \in \mathcal{Q}_i$ with $d(\sigma, \sigma') \geq (c-2)2^i$, let C_q be a cone with opening angle $2\pi/k$ and apex $q \in \sigma$ that intersects σ'. Then the cone obtained from C_q by doubling its opening angle contains σ'.*

Lemma 4 *Let C_q be a cone with apex q and opening angle $4\pi/k$. Suppose there are two points p and r in C_q with $(c+1)2^i \geq |rq| \geq |pq| \geq (c-2)2^i$. Then $|pr| \leq ((4\pi/k)(c+1) + 3)2^i$.*

Lemma 5 *Let $k \geq 14$ and let*

$$t = (1 + \sqrt{2 - 2\cos(4\pi/k)})/(2\cos(4\pi/k) - 1).$$

For any distinct points $p, q, r \in \mathbb{R}^2$ with $|rq| \leq |pq|$ and $\alpha = \angle pqr \in [0, 4\pi/k]$, we have $|pr| \leq |pq| - |rq|/t$.

We are now ready to prove that H is a t-spanner. This is done in a similar manner as for Yao graphs.

Lemma 6 *For any $t > 1$, there are constants c and k such that H is a t-spanner for G.*

Proof. We show by induction on the rank of the length of the edges in G that for each edge $\overrightarrow{pq}$ in G there is a p-q-path of length at most $t|pq|$ in H.

Consider the shortest edge $\overrightarrow{pq}$ of G. Let C_q be the cone at q that contains p. There is at least one pair in N that fulfills Def. 1(ii) for $\overrightarrow{pq}$. Among those, we pick the pair $(\sigma, \sigma') \in N$ with minimum diameter. Suppose that $q \in \sigma$, $p \in \sigma'$, and $\operatorname{diam}(\sigma) = \operatorname{diam}(\sigma') = 2^i$.

Since $\overrightarrow{pq}$ is the shortest edge, σ' contains only p (taking $c > 3$) and $m_{\sigma'} = p$, so $R = \{p\}$. Furthermore, since $p \in \sigma'$, we have $C_q \cap \sigma' \neq \emptyset$. Thus, if q is active in σ, then $q \in Q$, and we pick the edge $\overrightarrow{pq}$ for H (Alg. 1, line 8). Suppose not. Then we have picked an incoming edge $\overrightarrow{rq}$ for a smaller pair $(\bar{\sigma}, \bar{\sigma}') \in N$ with $\operatorname{diam}(\bar{\sigma}) \leq 2^{i-1}$. By Def. 1(i), $|rq| \leq (c+1)2^i$. Also by Def. 1(i) we have $|pq| \geq (c-2)2^i$ and since $|rq| \geq |pq|$, we have $(c+1)2^i \geq |rq| \geq |pq| \geq (c-2)2^i$. By Lemma 3, $\bar{\sigma}'$ (and thus r) is contained in the cone C_q^2 obtained from C_q by doubling its angle to $4\pi/k$. Using Lemma 4 with C_q^2, we see that $|pr| \leq ((4\pi/k)(c+1)+3)2^i$. Since for $c, k \geq 14$ we have $(4\pi/k)(c+1)+3 < c-2$, this would mean that $|pr| < |pq| \leq r_p$. Thus, $\overrightarrow{pr}$ would be an edge of G that is strictly shorter than $\overrightarrow{pq}$, despite our choice of $\overrightarrow{pq}$. Hence, when processing σ, we will discover $\overrightarrow{pq}$.

For the inductive step, consider an edge $\overrightarrow{pq}$ and the cone C_q containing p. Again, let $(\sigma, \sigma') \in N$ be the smallest pair of cells with $q \in \sigma$ and $p \in \sigma'$ that fulfill Def. 1(ii) and suppose $\operatorname{diam}(\sigma) = 2^i$. We have $C_q \cap \sigma' \neq \emptyset$, and we distinguish two cases.

Case 1: q is active. Then $q \in Q$ and Def. 1(i) guarantees that Alg. 1 obtains an incoming edge $\overrightarrow{rq}$ for q with $r \in \sigma'$. If $r = p$, we are done, so suppose $r \neq p$. Since $|pr| \leq 2^i$, by induction there is a path from p to r in H of length at most $t2^i$. Using the triangle inequality, we estimate the distance $d(p,q)$ in H by

$$d(p,q) \leq t2^i + |rq| \leq t2^i + |pq| + 2^i = |pq| + (1+t)2^i.$$

For c large enough the bound $|pq| > (c-2)2^i$ gives

$$|pq| + (1+t)2^i \leq (1 + (1+t)/(c-2))|pq| \leq t|pq|.$$

Case 2: q is inactive. There is an edge $\overrightarrow{rq}$ that was selected due to a pair $(\bar{\sigma}, \bar{\sigma}') \in N$ with $q \in \bar{\sigma}$, $r \in \bar{\sigma}'$ and $\operatorname{diam}(\bar{\sigma}) \leq 2^{i-1}$. By Lemma 3, p and r are contained in the cone C_q^2 with opening angle $4\pi/k$. We distinguish two subcases.

First, suppose that $|rq| \geq |pq|$. Then, since $(c+1)2^i \geq |rq| \geq |pq| \geq (c-2)2^i$, Lemma 4 implies that $r \in D(p)$, so $\overrightarrow{pr}$ is an edge of G of length at most $((4\pi/k)(c+1)+3)2^i$. Thus, we can bound $d(p,q)$ by

$$\begin{aligned} &t|pr| + |rq| \leq t\left(4\pi(c+1)/k + 3\right)2^i + (c+1)2^i \\ &= \left(t\left(4\pi(c+1)/k + 3\right) + c + 1\right)|pq|/(c-2) \leq |pq|t, \end{aligned}$$

for $c, k = \Theta(t/(t-1))$. Here we used the fact that $|rq| \leq (c+1)2^i$ and that $2^i \leq |pq|/(c-2)$.

Second, suppose $|rq| < |pq|$. By Lemma 5, we get $|pr| \leq |pq| - |rq|/t$. Thus, $\overrightarrow{pr}$ is an edge of G, and

$$d(p,q) \leq t|pr| + |rq| \leq t(|pq| - |rq|/t) + |rq| = t|pq|,$$

where the first inequality is by induction. □

Finding the Decomposition. We show how to find the decomposition for G as in Def. 1. Let $c > 3$ and scale P so that the closest pair in P has distance c. A *quadtree* for P is a rooted tree T where each internal node has degree four. Each node v of T has an associated cell σ_v from a grid $\mathcal{Q}_i$, $i \geq 0$, and we say that v has *level* i. If v is internal, the cells of its four children partition σ_v into four congruent squares with half the diameter of σ_v. We compute a quadtree T for P and use it to find a c-separated annulus decomposition.

We construct T level-wise. To begin, we take the smallest integer L such that there is a cell $\sigma \in \mathcal{Q}_L$ that contains P. Since c is constant and since P has spread Φ, the scaled point set has diameter $c\Phi$, and $L = O(\log \Phi)$ (possibly after shifting P). We create the root v and set $\sigma_v = \sigma$. This gives level L. To construct level $i-1$ from level i, we do the following for each level-i-node v whose cell σ_v is non-empty: we take the four cells of $\mathcal{Q}_{i-1}$ that partition σ_v and create four children $w_1, \ldots, w_4$ of v. To each of $w_1, \ldots, w_4$ we assign one of the four cells. We stop at level 0. The scaling of P ensures that a cell of level 0 contains at most one point and has diameter 1.

We now set $\mathcal{Q} = \{\sigma_v \mid v \in T\}$. We let $(\sigma_v, \sigma_w) \in N$ if v and w have the same level and $d(\sigma_v, \sigma_w) \in [(c-2)\operatorname{diam}(\sigma_v), 2c\operatorname{diam}(\sigma_v))$. As R_{σ_v}, we take all $p \in \sigma_v \cap P$ with $r_p \in [(c-2)\operatorname{diam}(\sigma_v), 2(c+1)\operatorname{diam}(\sigma_v)]$.

Lemma 7 *The set $\mathcal{Q}$ with N and R_σ as above is a c-separated annulus decomposition with $|\mathcal{Q}| = O(n)$.*

Proof. Since T has $O(n)$ nodes, we have $|\mathcal{Q}| = O(n)$. Property (i) of Def. 1 follows by construction. For Property (ii), let $\overrightarrow{pq}$ be an edge of G and let $i \in \mathbb{N}_0$ such that $|pq| \in [c2^i, c2^{i+1})$. Let $\sigma, \sigma' \in \mathcal{Q}_i$ with $p \in \sigma$ and $q \in \sigma'$. By construction, these cells are assigned to nodes of T and thus $\sigma, \sigma' \in \mathcal{Q}$. Since $\operatorname{diam}(\sigma) = \operatorname{diam}(\sigma') = 2^i$, we have $(c-2)2^i \leq d(\sigma, \sigma') \leq |pq| < c2^{i+1}$, so $(\sigma, \sigma') \in N$. Since $\overrightarrow{pq}$ is an edge of G, we have $r_p \geq |pq| \geq c2^i$. If $r_p \leq (c+1)2^{i+1}$, then $p \in R_\sigma$. Otherwise, $r_{m_\sigma} \geq r_p > (c+1)2^{i+1}$, and $D(m_\sigma)$ contains σ' and also q. □

Running Time. Considering the cells of $\mathcal{Q}$ in increasing order in Alg. 1 constitutes a level-order traversal of T starting from level 0. Fix a cell σ_v of a node v of T. We can sort $\sigma_v \cap P$ in the preprocess step (line 5) by merging the sorted lists of v's children. This takes $O(n)$ time per level and $O(n \log \Phi)$ time in total. Now we bound the time for edge selection.

Lemma 8 *Let Q, R as in Alg.1, line 8 with $|Q| = n$ and $|R| = m$. For each $q \in Q$ we can find an $r \in R$ with $q \in D(r)$, if such r exists, in time $O(m \log m + n)$.*

Proof. Q and R are separated by one the supporting lines ℓ of the cell σ that contains Q. Since σ is axis-aligned, Q is sorted along ℓ in the preprocess step. Consider a coordinate system with x-axis ℓ. The lower envelope E of the disks of R and ℓ has $O(m)$ arcs, can

be computed in $O(m \log m)$ time and is monotone in ℓ direction: since ℓ separates R and Q, each arc and ℓ can be seen as a function of x, and E is the pointwise minimum of these functions (cf. Fig. 2). Let S be the points on E where the arcs change. We merge Q and S in time $O(m+n)$, and we sweep over $Q \cup S$ in x-direction to compute the point-disk incidences for Q and R. We initialize D as the disk of the first arc and q as the first point of Q. Whenever we reach a point $p \in S \cup Q$, we update D or q, depending on whether $p \in S$ or $p \in Q$. In the former case, we set D to the disk of the new arc. In the latter case, we first set $q = p$, and then we check if $q \in D$. If so, we assign D to q. This sweep takes $O(m+n)$ time. Since the lower envelope is monotone, it is enough to check for each $q \in Q$ only the arc intersected by the line through q orthogonal to ℓ. □

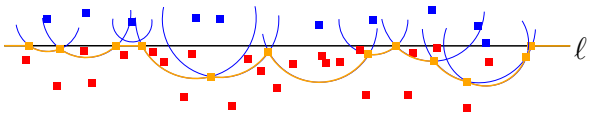


Figure 2: The lower envelope and S (orange), the points Q (red), and R (blue).

The next lemma states the running time of Alg. 1. Due to space reasons, we omit the proof. The main idea is that the running time is dominated by the edge selection step. By the choice of R_σ, each point in P participates in $O(1)$ edge selections as a disk center, at a cost of $O(\log n)$ per disk center (by Lemma 8), and in $O(\log \Phi)$ edge selections as a point in Q, at $O(1)$ cost per point (by Lemma 8). Thm. 1 follows by Lemmas 6 and 9.

Lemma 9 *The construction of the spanner H of G takes $O(n(\log \Phi + \log n))$ time.*

3 From Bounded Spread to Bounded Radii

To get Theorem 2, we extend Alg. 1 from §2. We show that the spread is irrelevant: points that are close together form cliques in G and can be handled through classic spanners; points that are far away from each other form pairwise independent components.

Given t, we pick the separation parameter c large enough. We scale P such that the smallest radius is c. Let $M = O(\Psi)$ be the largest radius. We partition P into independent components. For this, we put around each $p \in P$ an axis-parallel square of side length $2M$. The connected components of the intersection graph of the squares give the sets. We state this in the next lemma. whose proof we omit.

Lemma 10 *In $O(n \log n)$ time, we can partition P into sets $P_1, \ldots, P_\ell$ of diameter $O(n\Psi)$ so that for $i \neq j$, no point in P_i can reach a point in P_j in G.*

By Lemma 10, we may assume that P has diameter $O(n\Psi)$. As in §2, we compute a quadtree T for P with L levels and $L = O(\log(n\Psi))$. Unlike in §2, T does not directly yield a c-separated annulus decomposition for G. Def. 1(ii) does not hold, since there may be edges in G that do not go between neighboring cells. These are the *short edges*.

First, we handle very short edges: let v be a level 0 node of T with associated cell $\sigma_v \in \mathcal{Q}_0$. Let $Q \subseteq P$ be the points in cells of $\mathcal{Q}_0$ with distance at most $c/2-3$ from σ_v. Since any two points in Q have distance at most c, Q is a clique in G. We compute a (classic) t-spanner for Q in $O(|Q| \log |Q|)$ time [4]. Since any $p \in P$ is in $O(c^2)$ such spanners, we generate $O(n)$ edges in total and require $O(n \log n)$ running time.

Second, we handle not quite so short edges: for each $q \in P$, let v be the level 0 node of T whose cell σ_v contains q. For any non-empty $\sigma' \in \mathcal{Q}_0$ with $d(\sigma_v, \sigma') \in (c/2-3, c-2)$, we take an arbitrary point $r \in \sigma' \cap P$ and add the edge $\overrightarrow{rq}$ to our spanner. All these edges have length at most c and thus are edges in G. This takes $O(n)$ time and creates $O(n)$ edges.

Finally, we handle the remaining edges: we mark all points in P as active, and we run Alg. 1 from §2 for the cells of T. Call the resulting graph H.

As in Lemma 6, we can show inductively that each edge of G is approximated in H. The differences are in the base case: if the shortest edge in G is very short, the classic spanner does the job. If it is a not quite so short, a calculation as in Lemma 6 shows that we pick it. Otherwise, the base case is as in Lemma 6.

Lemma 11 *For any $t > 1$, there are constants c,k such that the graph H as above is a t-spanner for G.*

Thm. 2 follows as in §2. The running time analysis goes as in Lemma 9, but the quadtree has $O(\log n + \log \Psi)$ levels.

Acknowledgments. We like to thank Paz Carmi for valuable comments.

References

[1] P. Bose, M. Damian, K. Douïeb, J. O'Rourke, B. Seamone, M. H. M. Smid, and S. Wuhrer. $\pi/2$-Angle Yao Graphs are Spanners. *IJCGA*, 22(1):61–82, 2012.

[2] P. Carmi. personal communication, 2014.

[3] M. Fürer and S. P. Kasiviswanathan. Spanners for geometric intersection graphs with applications. *J. Comput. Geom.*, 3(1):31–64, 2012.

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

[5] D. Peleg and L. Roditty. Localized spanner construction for ad hoc networks with variable transmission range. *TOSN*, 7(3), 2010.

[6] A. C.-C. Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

Constrained Generalized Delaunay Graphs Are Plane Spanners

Prosenjit Bose* Jean-Lou De Carufel* André van Renssen†,‡

Abstract

We look at generalized Delaunay graphs in the constrained setting by introducing line segments which the edges of the graph are not allowed to cross. Given an arbitrary convex shape C, an unconstrained Delaunay graph is constructed by adding an edge between two vertices p and q if and only if there exists a homothet of C with p and q on its boundary that does not contain any other vertices. We show that, regardless of the convex shape used to construct the constrained Delaunay graph, there exists a constant t such that it is a plane t-spanner.

1 Introduction

A geometric graph G is a graph whose vertices are points in the plane and whose edges are line segments between pairs of points. Every edge is weighted by the Euclidean distance between its endpoints. The distance between two vertices u and v in G, denoted by $\delta_G(u, v)$ is defined as the sum of the weights of the edges along the shortest path between u and v in G. A subgraph H of G is a t-spanner of G (for $t \geq 1$) if for each pair of vertices u and v, $\delta_H(u, v) \leq t \cdot \delta_G(u, v)$. The smallest value t for which H is a t-spanner is the *spanning ratio* or *stretch factor* of H. The graph G is referred to as the *underlying graph* of H. The spanning properties of various geometric graphs have been studied extensively in the literature (see [5] for a comprehensive overview of the topic).

Most of the research has focused on constructing spanners where the underlying graph is the complete Euclidean geometric graph. We study this problem in a more general setting with the introduction of line segment *constraints*. Specifically, let P be a set of points in the plane and let S be a set of line segments with endpoints in P, with no two line segments intersecting properly. The line segments of S are called *constraints*. Two vertices u and v can *see each other* or *are visible to each other* if and only if either the line segment uv does not properly intersect any constraint or uv is itself a constraint. If two vertices u and v can see each other, the line segment uv is a *visibility edge*. The *visibility graph* of P with respect to a set of constraints S, denoted $\mathrm{Vis}(P, S)$, has P as vertex set and all visibility edges as edge set. In other words, it is the complete graph on P minus all edges that properly intersect one or more constraints in S.

This setting has been studied extensively within the context of motion planning amid obstacles. Clarkson [6] was one of the first to study this problem and showed how to construct a linear-sized $(1 + \epsilon)$-spanner of $\mathrm{Vis}(P, S)$. Subsequently, Das [7] showed how to construct a spanner of $\mathrm{Vis}(P, S)$ with constant spanning ratio and constant degree. Bose and Keil [3] showed that the Constrained Delaunay Triangulation is a $4\pi\sqrt{3}/9 \approx 2.419$-spanner of $\mathrm{Vis}(P, S)$. The constrained Delaunay graph where the empty convex shape is an equilateral triangle was shown to be a 2-spanner of $\mathrm{Vis}(P, S)$ [2].

We show that the constrained generalized Delaunay graph G is a spanner whose spanning ratio depends solely on the empty convex shape C used to create it: We show that G satisfies the α_C-diamond property and the visible-pair κ_C-spanner property (defined in Section 4), which implies that it is a t-spanner for:

$$t = \begin{cases} 2\kappa_C \cdot \max\left(\frac{3}{\sin(\alpha_C/2)}, \kappa_C\right), & \textit{for triangulations} \\ 2\kappa_C^2 \cdot \max\left(\frac{3}{\sin(\alpha_C/2)}, \kappa_C\right), & \textit{otherwise.} \end{cases}$$

2 Preliminaries

Throughout this paper, we fix a convex shape C. We assume without loss of generality that the origin lies in the interior of C. A *homothet* of C is obtained by scaling C with respect to the origin, followed by a translation.

Given a set of vertices P and a set of constraints S, we now define the constrained generalized Delaunay graph. Given any two visible vertices p and q, let $C(p, q)$ be any homothet of C with p and q on its boundary. The constrained generalized Delaunay graph contains an edge between p and q if and only if there exists a $C(p, q)$ such that $C(p, q)$ does not contain any vertices visible to both p and q. Note that this implies that constraints are *not* necessarily edges of the constrained generalized Delaunay graph. Joe and Wang showed that the constrained generalized Delaunay graph is not necessarily the dual of the constrained Voronoi diagram [9].

*School of Computer Science, Carleton University, Ottawa, Canada. Research supported in part by FQRNT, NSERC, and Carleton University's President's 2010 Doctoral Fellowship. Email: jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca.

†National Institute of Informatics (Nii), Tokyo, Japan. andre@nii.ac.jp

‡JST, ERATO, Kawarabayashi Large Graph Project

2.1 Auxiliary Lemmas

Next, we present three auxiliary lemmas that are needed to prove our main results. First, we reformulate a lemma that appears in [10].

Lemma 1 *Let C be a convex closed curve in the plane. The intersection of two distinct homothets of C is the union of two sets, each of which is either a segment, a single point, or empty.*

We continue with a property of visibility graph from [2]. To avoid confusion, we define that we call a region *empty* if it does not contain any vertex of P.

Lemma 2 *Let u, v, and w be three arbitrary points in the plane such that uw and vw are visibility edges and w is not the endpoint of a constraint intersecting the interior of triangle uvw. Then there exists a convex chain of visibility edges from u to v in triangle uvw, such that the polygon defined by uw, wv and the convex chain is empty and does not contain any constraints.*

Given two vertices p and q that can see each other and a convex polygon $C(p,q)$ with p and q on its boundary, we look at the constraints that have p as an endpoint and the edge(s) of $C(p,q)$ on which p lies and extend them to half-lines that have p as an endpoint (see Figure 1). Given the cyclic order of these half-lines around p and the line segment pq, we define the clockwise neighbor of pq to be the half-line that minimizes the strictly positive clockwise angle with pq. Analogously, we define the counterclockwise neighbor of pq to be the half-line that minimizes the strictly positive counterclockwise angle with pq. We define the *cone* C_q^p that contains q to be the region between the clockwise and counterclockwise neighbor of p. Finally, let $C(p,q)_q^p$, the *region of $C(p,q)$ that contains q with respect to p*, be the intersection of $C(p,q)$ and C_q^p.

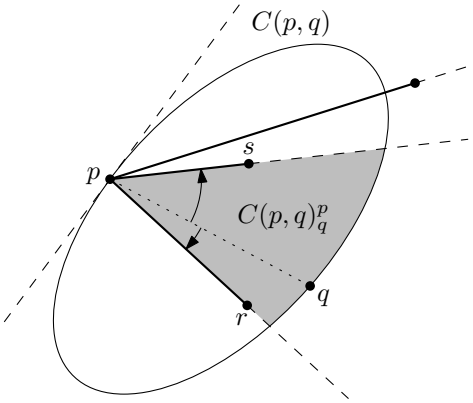


Figure 1: Defining the region of $C(p,q)$ that contains q with respect to p.

Lemma 3 *Let p and q be two vertices that can see each other and let $C(p,q)$ be any convex polygon with p and q on its boundary. If $C(p,q)$ contains a vertex x in $C(p,q)_q^p$ that is visible to p, then $C(p,q)$ contains a vertex y that is visible to both p and q and triangle pyq is empty.*

Proof. We have two visibility edges, namely pq and px. Since x lies in $C(p,q)_q^p$, p is not the endpoint of a constraint such that q and x lie on opposite sides of the line through this constraint. Hence, we can apply Lemma 2 and we obtain a convex chain of visibility edges from x to q and the polygon defined by pq, px and the convex chain is empty and does not contain any constraints. Furthermore, since the convex chain is contained in pxq, which is contained in $C(p,q)$, every vertex along the convex chain is contained in $C(p,q)$.

Let y be the neighbor of q along this convex chain. Hence, y is visible to q and contained in $C(p,q)$. Also, p can see y, since the line segment py is contained in the polygon defined by pq, px and the convex chain, which is empty and does not contain any constraints. This implies that triangle pyq is empty. □

3 Planarity

Before we show that the constrained generalized Delaunay graph is a spanner, we show that it is plane.

Lemma 4 *Let pq be an edge of the constrained generalized Delaunay graph. The line segment pq does not contain any vertices other than p and q.*

Lemma 5 *The constrained generalized Delaunay graph is plane.*

Proof. We prove this by contradiction, so assume that there exist two edges pq and rs that intersect properly, i.e. not at their endpoints. It follows from Lemma 4 that neither p nor q lies on rs and that neither r nor s lies on pq. Since pq is contained in $C(p,q)$ and rs is contained in $C(r,s)$, $C(p,q)$ and $C(r,s)$ intersect.

We first show that this implies that $p \in C(r,s)$, $q \in C(r,s)$, $r \in C(p,q)$, or $s \in C(p,q)$. If either $p \in C(r,s)$ or $q \in C(r,s)$, we are done, so assume that neither p nor q lies in $C(r,s)$. Lemma 1 states that $C(p,q)$ and $C(r,s)$ intersect each other at most twice. These intersections split the boundary of $C(p,q)$ into two parts: one that is contained in $C(r,s)$ and one that is not. Since $p \notin C(r,s)$ and $q \notin C(r,s)$, p and q lie on the arc of $C(p,q)$ that is not contained in $C(r,s)$ (see Figure 2). However, pq intersects $C(r,s)$, since otherwise pq cannot intersect rs. Let x and y be the two intersections of pq with the boundary of $C(r,s)$ (if the boundary of $C(r,s)$ is parallel to pq, x and y are the two endpoints of the interval of this intersection). We note that x and y split $C(r,s)$ into two parts, one of which is contained in $C(p,q)$, and

that r and s cannot lie on the same part. In particular, one of r and s lies on the part that is contained in $C(p,q)$, proving that $r \in C(p,q)$, or $s \in C(p,q)$.

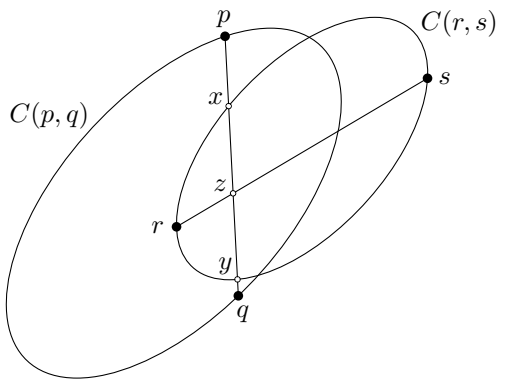


Figure 2: $C(p,q)$ and $C(r,s)$ intersect and p and q intersect $C(r,s)$ at x and y.

In the remainder of the proof, we assume without loss of generality that $r \in C(p,q)$ (see Figure 2). Let z be the intersection of pq and rs. Hence, z can see both p and r. Also, z is not the endpoint of a constraint intersecting the interior of triangle pzr. Therefore, it follows from Lemma 2 that there exists a convex chain of visibility edges from p to r. Let v be the neighbor of p along this convex chain. Since v is part of the convex chain, which is contained in pzr, which in turn is contained in $C(p,q)$, it follows that v is a vertex visible to p contained in $C(p,q)$. Furthermore, since the polygon defined by pz, zr and the convex chain does not contain any constraints, v lies in $C(p,q)^p_q$. Thus, it follows from Lemma 3 that there exists a vertex in $C(p,q)$ that is visible to both p and q, contradicting that pq is an edge of the constrained generalized Delaunay graph. □

4 Spanning Ratio

Let x and y be two distinct points on the boundary ∂C of C. These two points split ∂C into two parts. For each of these parts, there exists an isosceles triangle with base xy such that the third vertex lies on that part of ∂C. We denote the base angles of these two triangles by $\alpha_{x,y}$ and $\alpha'_{x,y}$. We define α_C as follows: $\alpha_C = \min\{\max(\alpha_{x,y}, \alpha'_{x,y}) : x, y \in \partial C, x \neq y\}$.

Given a graph G and an angle $0 < \alpha < \pi/2$, we say that an edge pq of G satisfies the *α-diamond property*, when at least one of the two isosceles triangles with base pq and base angle α does not contain any vertex visible to both p and q. A graph G satisfies the α-diamond property when all of its edges satisfy this property [8]. Using a proof analogous to the proof of Lemma 3 in [1], we can show the following:

Lemma 6 *Let C be any convex polygon. The constrained generalized Delaunay graph satisfies the α_C-diamond property.*

Next, let O be a point in the interior of C and let x and y be two distinct points on ∂C, such that x, y, and O are collinear. Again, x and y split ∂C into two parts. Let $\ell_{x,y}$ and $\ell'_{x,y}$ denote the length of these two parts. We define $\kappa_{C,O}$ as follows: $\kappa_{C,O} = \max\{\max(\ell_{x,y}, \ell'_{x,y})/|xy| : x, y \in \partial C, x \neq y,$ and $x, y,$ and O are collinear$\}$.

We note that the constrained generalized Delaunay graph does not depend on the location of O inside C. Therefore, we define κ_C as follows: $\kappa_C = \min\{\kappa_{C,O} :$ O is in the interior of $C\}$.

Given a constrained generalized Delaunay graph G, let p and q be two vertices on the boundary of a face f of the constrained generalized Delaunay graph, such that p can see q and the line segment pq does not intersect the exterior of f. If for every such pair p and q on every face f, there exists a path in G of length at most $\kappa \cdot |pq|$, then G satisfies the *visible-pair κ-spanner property*. Given a path between two vertices p and q, we call the path *one-sided*, if all vertices lie above pq or all vertices lie below pq.

Let a set of $k+1$ points $v_1, ..., v_{k+1}$ be given, such that all points lie on one side of the line through v_1 and v_{k+1}. For ease of exposition, assume the line through v_1 and v_{k+1} is the x-axis and all points lie on or above this line. We consider only point sets for which there exists $C_1, ..., C_k$, a set of homothets of C, such that the center of each homothet lies on the x-axis, C_i has v_i and v_{i+1} on its boundary, and no C_i contains any vertices other than v_i and v_{i+1}. Let ∂C be the boundary of C above the x-axis and let $\partial(v_i, v_{i+1})$ be the part of the boundary of C_i between v_i and v_{i+1} that lies above the x-axis.

Lemma 7 *Let $C(v_1, v_{k+1})$ be the homothet with v_1 and v_{k+1} on its boundary and its center on the x-axis. It holds that $\sum_{i=1}^{k} |\partial(v_i, v_{i+1})| \leq |\partial C(v_1, v_{k+1})|$.*

Lemma 8 *The constrained generalized Delaunay graph satisfies the visible-pair κ_C-spanner property.*

Proof. Let p and q be two vertices on the boundary of a face f, such that p can see q and the line segment pq does not intersect the exterior of f. Assume without loss of generality that pq lies on the x-axis. Let $C(p,q)$ be the homothet of C with p and q on its boundary and its center on pq. We aim to show that there exists a path between p and q of length at most $\kappa_C \cdot |pq|$. Since κ_C is at least $|\partial C(p,q)|/|pq|$, showing that there exists a path between p and q of length at most $|\partial C(p,q)|$ completes the proof. If pq is an edge of the constrained generalized Delaunay graph, this follows from the triangle inequality.

We grow a homothet C' with its center on pq by moving its center from p to q, while maintaining that p lies on the boundary of C'. Let v_1 be the first vertex hit by C' that is visible to p and lies in $C(p,q)^p_q$. We assume without loss of generality that v_1 lies above pq.

Since v_1 is the first vertex satisfying these conditions, pv_1 is either an edge or a constraint: Since v_1 is the first visible vertex we hit in $C(p,q)^p_q$, we have that $C(p,q)^p_q \cap C'$ contains no vertices visible to p. Hence, there is no vertex visible to both p and v_1. Therefore, Lemma 3 implies that $C(p,q)^p_q \cap C'$ does not contain any vertices visible to v_1. Finally, if pv_1 is not a constraint, $C(p,q)^p_q \cap C'$ contains the region that is visible to both p and v_1. Hence, if pv_1 is not a constraint, the region that is visible to both p and v_1 does not contain any vertices and pv_1 is an edge.

We continue constructing a sequence of vertices $p, v_1, v_2, ..., v_k, q$ until we hit q by moving the center of C' along pq towards q and each time we hit a vertex v_i, we require that it lies on the boundary of C' until we hit the next vertex v_{i+1} that is visible to v_i and v_i is not the endpoint of a constraint that intersects the interior of triangle $v_{i-1}v_iv_{i+1}$. Since v_{i+1} is the first vertex satisfying these conditions starting from v_i, v_iv_{i+1} is either an edge or a constraint. This in turn implies that these vertices all lie above pq, since pq is visible and does not intersect the exterior of f.

Unfortunately, we cannot assume that there exists an edge between every pair of consecutive vertices: If v_iv_{i+1} is a constraint, there can be vertices visible to both v_i and v_{i+1} above the constraint. For the pairs of vertices v_i, v_{i+1} that do not form an edge, we refine the construction of the sequence between them: We start with C' such that it does not cross v_iv_{i+1} and v_i lies on its boundary. We construct a sequence of vertices from v_i to v_{i+1} by moving the center of C' along pq towards q. For the first vertex we hit, we require that it is visible to v_i and lies in $C'^{v_i}_{v_{i+1}}$.

We continue moving the center of C' along pq towards q, but we now maintain that v'_i lies on the boundary of C'. Each time we hit a vertex v_j, we require that it lies on the boundary of C' until we hit the next vertex v'_{j+1} that is visible to v_j and v_j is not the endpoint of a constraint that intersects the interior of $v_{j-1}v_jv_{j+1}$. In other words, we construct a more fine-grained sequence when consecutive vertices define a constraint and there is no edge between them. Note that we may need to repeat this process a number of times, since there need not be edges between the vertices of the finer grained sequence either.

This way, we obtain a path $p, v'_1, v'_2, ..., v'_l, q$ from p to q that lies above pq. Since C is convex, we can upper bound the length of each edge v_iv_{i+1} by the part of the boundary the homothet with v_i and v_{i+1} on its boundary and its center on pq, that does not intersect pq. Hence, the total length of the path is upper bounded by the length of the union of these partial boundaries. By construction, none of the homothets corresponding to consecutive vertices along the path contain any of the other vertices along the path. Hence, we can apply Lemma 7 and it follows that the total length of the path is at most $|\partial C(p,q)|$. □

Das and Joseph [8] showed that any plane graph that satisfies the diamond property and the good polygon property (similar to the visible-pair κ-spanner property) is a spanner. Subsequently, Bose *et al.* [4] improved slightly on the spanning ratio. They showed that a geometric (constrained) graph G is a spanner of the visibility graph when it: (1) is plane, (2) satisfies the α-diamond property, (3) the spanning ratio of any one-sided path in G is at most κ, and (4) satisfies the visible-pair κ'-spanner property. In particular, G is a t-spanner for $t = 2\kappa\kappa' \cdot \max(3/\sin(\alpha/2), \kappa)$.

It follows from Lemmas 5, 6, and 8 that the constrained generalized Delaunay graph satisfies these four properties. Thus, we obtain the following theorem:

Theorem 9 *The constrained generalized Delaunay graph G is a t-spanner for*

$$t = \begin{cases} 2\kappa_C \cdot \max\left(\frac{3}{\sin(\alpha_C/2)}, \kappa_C\right), & \text{for triangulations} \\ 2\kappa_C^2 \cdot \max\left(\frac{3}{\sin(\alpha_C/2)}, \kappa_C\right), & \text{otherwise.} \end{cases}$$

References

[1] P. Bose, P. Carmi, S. Collette, and M. Smid. On the stretch factor of convex Delaunay graphs. *JoCG*, 1(1):41–56, 2010.

[2] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. On plane constrained bounded-degree spanners. In *LATIN*, pages 85–96, 2012.

[3] P. Bose and J. M. Keil. On the stretch factor of the constrained Delaunay triangulation. In *ISVD*, pages 25–31, 2006.

[4] P. Bose, A. Lee, and M. Smid. On generalized diamond spanners. In *WADS 2007*, pages 325–336, 2007.

[5] P. Bose and M. Smid. On plane geometric spanners: A survey and open problems. *CGTA*, 46(7):818–830, 2013.

[6] K. Clarkson. Approximation algorithms for shortest path motion planning. In *STOC*, pages 56–65, 1987.

[7] G. Das. The visibility graph contains a bounded-degree spanner. In *CCCG*, pages 70–75, 1997.

[8] G. Das and D. Joseph. Which triangulations approximate the complete graph? In *Proc. Int. Symp. on Optimal Algorithms*, pages 168–192, 1989.

[9] B. Joe and C. A. Wang. Duality of constrained Voronoi diagrams and Delaunay triangulations. *Algorithmica*, 9(2):142–155, 1993.

[10] K. Swanepoel. Helly-type theorems for homothets of planar convex curves. *Proc. American Mathematical Society*, 131(3):921–932, 2003.

Two-Level Rectilinear Steiner Trees

Stephan Held* Nicolas Kämmerling*

Abstract

Given a set P of terminals in the plane and a partition of P into k subsets $P_1, \ldots, P_k$, a two-level rectilinear Steiner tree consists of a rectilinear Steiner tree T_i connecting the terminals in each set P_i $(i = 1, \ldots, k)$ and a top-level tree T_{top} connecting the trees $T_1, \ldots, T_k$. The goal is to minimize the total length of all trees. This problem arises naturally in the design of low-power physical implementations of parity functions on a computer chip.

For bounded k we present a polynomial time approximation scheme (PTAS) that is based on Arora's PTAS for rectilinear Steiner trees after lifting each partition into an extra dimension.

For the general case we propose an algorithm that predetermines a connection point for each T_i and T_{top} $(i = 1, \ldots, k)$. Then, we apply any approximation algorithm for minimum rectilinear Steiner trees in the plane to compute each T_i and T_{top} independently.

This gives us a 2.37-factor approximation with a running time of $\mathcal{O}(|P| \log |P|)$ suitable for fast practical computations. The approximation factor reduces to 1.63 by applying Arora's approximation scheme in the plane.

1 Introduction

We consider the *two-level rectilinear Steiner tree problem* (R2STP) that arises from an application in VLSI design. Consider the computation of a parity function of k input bits using 2-input XOR-gates. Due to the symmetry, associativity, and commutativity of the XOR function, this can be realized by an arbitrary binary tree with k leaves, rooted at the output simply by inserting an XOR-gate at every internal vertex [11, 12]. Throughout this paper we consider the parity function as a placeholder for any fan-in function of the type $x_1 \circ x_2 \circ \cdots \circ x_k$, where $\circ$ is a symmetric, associative, and commutative 2-input operator, i. e. $\circ \in \{\oplus, \vee, \wedge\}$.

On a chip such a tree has to be embedded into the plane and all connections must be realized by rectilinear segments. If each input and the output are single points on the chip, a realization of minimum length and thus power consumption is given by a minimum length rectilinear Steiner tree. This is a tree connecting the inputs and the output by horizontal and vertical line segments using additional so-called Steiner vertices to achieve a shorter length than a minimum spanning tree. At each Steiner vertex of degree three an XOR-gate is placed. Higher degree vertices can be dissolved into degree three vertices sharing their position. Figure 1 shows an example of an embedded parity function on the left.

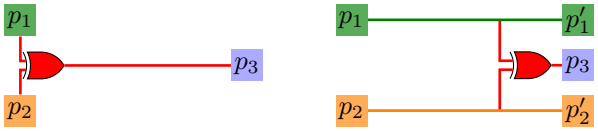


Figure 1: On the left, we have two inputs p_1 and p_2 and a single output p_3. The XOR-gate should be placed at the median of the three terminals. If the inputs have the side outputs p'_1 and p'_2, the XOR-gate should be placed at p_3, saving the horizontal length.

In practice input signals may be needed for other computations on the chip and thus delivered to other side outputs. Similarly, the result may have to be delivered to multiple output terminals. Thus, each input and its successors and the output terminals must be connected by separate Steiner trees as well. These trees are then connected by a top-level Steiner tree into which the XOR-gates will be inserted. Considering the additional terminals allows to construct a potentially shorter Steiner tree as shown in Figure 1 on the right. Algorithms ignoring the side outputs cannot guarantee an approximation factor better than two, as we will see in Section 2.

This motivates the definition of the *minimum two-level rectilinear Steiner tree problem*, where we are given a set $P \subset \mathbb{R}^2$ of n terminals and a partition of P into k subsets $P_1, \ldots, P_k$.

A *two-level rectilinear Steiner tree* $T = (T_{top}, T_1, \ldots, T_k)$ consists of a Steiner tree T_i for each $i \in \{1, \ldots, k\}$ connecting the terminals in P_i and a (group) Steiner tree T_{top} connecting the embedded trees $\{T_1, \ldots, T_k\}$. We call T_{top} the *top-level tree*. Note that all trees are allowed to cross. The objective is to minimize the total length of all trees

$$l(T) := \sum_{i=1}^{k} l(T_i) + l(T_{top}),$$

where $l(T') := \sum_{\{x,y\} \in E(T')} \|x - y\|_1$ is the ℓ_1-length of a Steiner tree T'.

*Research Institute for Discrete Mathematics, University of Bonn,
held@or.uni-bonn.de, kaemmerling@or.uni-bonn.de.

For each $i \in \{1, \ldots, k\}$ the top-level tree and T_i intersect in at least one point. We can select one such point $q_i \in T_{top} \cap T_i$ and call it *connection point* for T_i and T_{top}. Then T_{top} is a Steiner tree for the terminals $\{q_1, \ldots, q_k\}$ and each T_i is a Steiner tree for $P_i \cup \{q_i\}$.

Obviously, this problem is *NP*-hard as it contains the minimum rectilinear Steiner tree problem in two ways: if $k = 1$ or if $|P_i| = 1$ for $i \in \{1, \ldots, k\}$.

Designing the top-level tree as a stand-alone problem is hard. If all subtrees T_i ($i \in \{1, \ldots, k\}$) are fixed, T_{top} cannot be approximated to arbitrary quality, as the group Steiner tree problem for connected groups in the Euclidean plane cannot be approximated within a factor of $(2 - \epsilon)$ [9]. However we are in a more lucky situation as we can tradeoff the lengths of bottom-level and top-level trees.

To the best of our knowledge the two-level rectilinear Steiner tree problem has not been considered before despite its practical importance [11, 12]. It is loosely related to the hierarchical network design problems [1, 5, 6] or multi-level facility location problems [3, 4]. However, those problems are structurally different, typically considering problems in graphs, and do not apply to our case.

In [11], ordinary rectilinear Steiner trees were used to build power efficient fan-in trees, when each input and the output consists of a single terminal. In practice designers are also interested in the depth of the constructed circuit [12]. However, for finding good power versus depth tradeoffs a better understanding of short solutions is an essential prerequisite and the aim of our work.

1.1 Our Contribution

In Section 2, we show that the naïve approach of picking a random terminal from each partition as a connection point to the top-level tree and building the bottom-level trees and top-level as separate instances gives a 2α-factor approximation, where α is the approximation factor of the used minimum Steiner tree algorithm. This works for arbitary metrics.

Then in Section 3 and 4, we focus on rectilinear instances. In Section 3 we show how to lift our instance into an equivalent $(2+k)$-dimensional rectilinear Steiner tree instance. If the number k of partitions is bounded by a constant, we obtain a PTAS by applying Arora's PTAS for rectilinear Steiner trees [2].

As our main result, we improve the approximation guarantee for unbounded k from $(2 + \epsilon)$ to 1.63 in Section 4. Using spanning tree heuristics, this approach turns also into a fast practical algorithm with running time $\mathcal{O}(n \log n)$ and approximation factor 2.37.

2 Simple Bottom-Up Construction

A simple bottom-up approach, which works for any metric space, is to compute a Steiner tree T_i for P_i ($i = 1, \ldots, k$). In each T_i we fix a connection point $q_i \in P_i$ arbitrarly, compute a Steiner tree T_{top} for $\{q_1, \ldots, q_k\}$, and return $T = (T_{top}, T_1, \ldots, T_k)$.

Figure 2: A tight example when choosing connection points as arbitary points of P_i.

Theorem 1 *The simple bottom-up approach is a 2α-factor approximation algorithm for the minimum two-level Steiner tree problem, if we use an α-factor approximation algorithm for the minimum Steiner tree problem as a subroutine.*

Proof. Let T be the two level Steiner tree computed by the simple bottom-up approach and let $T^\star = (T^\star_{top}, T^\star_1, \ldots, T^\star_k)$ be a minimum two-level Steiner tree. Let be $q^\star_i \in T^\star_{top} \cap T^\star_i$ the connection point of the optimum two-level Steiner tree. Since $T^\star_i$ is a Steiner tree on $\{q^\star_i\} \cup P_i$, we have $dist(q^\star_i, q_i) \leq l(T^\star_i)$. Thus,

$$l(T) \leq \alpha \cdot l(T^\star_{top}) + \alpha \sum_{i=1}^{k} dist(q^\star_i, q_i) + \sum_{i=1}^{k} \alpha \cdot l(T^\star_i)$$
$$\leq \alpha \cdot l(T^\star_{top}) + 2\alpha \sum_{i=1}^{k} l(T^\star_i) \;\leq\; 2\alpha \cdot l(T^\star).$$

□

Figure 2 shows that the factor $(2 + \epsilon)$ is sharp. For the instance $P_1 = \{(0, 0), (1, 0)\}$, $P_2 = \{(0, 0), (-1, 0)\}$ a minimum two-level rectilinear Steiner tree of length 2 is shown on the left with $l(T_{top}) = 0$. On the right, a bad choice of connection points and minimum Steiner trees T_{top}, T_1, and T_3 yield a total length of 4.

3 PTAS for a bounded number of partitions

We can reduce the two-level rectilinear Steiner tree problem in the plane to an ordinary rectilinear Steiner tree problem in a higher dimensional space, where we can apply Arora's PTAS [2].

The idea of the PTAS is to lift every subset $P_1, \ldots, P_k$ to an additional dimension. We assume $k > 1$. Otherwise the two-level Steiner tree problem is an ordinary Steiner tree problem. Let $P_1, \ldots, P_k \subset \mathbb{R}^2$ be the subsets of a two-level Steiner tree instance, we define a Steiner tree instance in $\mathbb{R}^{2+k}$. The set of terminals P' is comprised as follows.

For each original terminal $x \in P_i \subset \mathbb{R}^2$ ($i \in \{1, \ldots, k\}$), we add a terminal $x' := (x, K \cdot e_i) \in \mathbb{R}^{2+k}$, where $e_i \in \mathbb{R}^k$ is the unit vector with value one at the i-th coordinate and K is a large constant, e. g. we could choose K as $l(B(P))$. Now for $x \in P_h$ and $y \in P_i$ the distance of their high dimensional copies $x', y' \in P'$ is $\|x' - y'\|_1 = \|x - y\|_1 + 2K\|e_h - e_i\|_1 = \|x - y\|_1 + 2K\delta_{h,i}$,

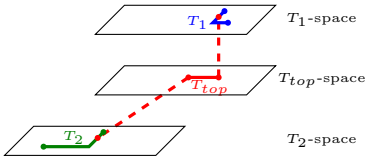


Figure 3: A flat Steiner Tree in a lifted instance.

where $\delta_{h,i}$ is one if $h = i$ and zero otherwise. An example of a lifted two-level Steiner tree is given in Figure 3.

A $(k+2)$-dimensional Steiner tree is called *flat* if all Steiner points have either the form $(x, 0) \in \mathbb{R}^{2+k}$ or $(x, K \cdot e_i) \in \mathbb{R}^{2+k}$, where $x \in \mathbb{R}^2$ and $e_i \in \mathbb{R}^k$ is a unit vector. The following Lemma has essentially been proven by Snyder [10], who shows that an optimum Steiner tree can be found in the d-dimensional Hanan grid [7].

Lemma 2 *A $(k+2)$-dimensional Steiner Tree T for P' of length $l(T)$ can be transformed in strongly polynomial time into a $(k+2)$-dimensional flat Steiner tree T' of length at most $l(T)$.*

The following lemma shows the equivalence between the original two-level rectilinear Steiner tree problem in the plane and the lifted regular rectilinear Steiner tree problem.

Lemma 3 *If $k > 1$, a two-level rectilinear Steiner tree T for $P_1, \ldots, P_k$ of length $l(T)$ can be transformed into a $(k+2)$-dimensional Steiner tree T' for P' of length at most $l(T) + kK$ and vice versa.*

Proof. We only sketch the proof, details can be found in [8]. We get T' by connecting the lifted components of T at its connection points by an edge of length K. Conversely, we flat T by applying Lemma 2 and replacing edges, s. t. for each $j \in \{3, \ldots, k+2\}$, T' contains at most one edge in direction j, without increasing the length of the tree. Projecting the edges onto the first two coordinates we obtain a feasible two-level Steiner tree of length at most $l(T') - kK$. □

Theorem 4 *For bounded k there is a PTAS for the two-level rectilinear Steiner tree problem.*

Proof. Choose $K = l(B(P))$ and for $\epsilon > 0$ set $\epsilon' := \frac{1}{k+1}\epsilon$. Then compute an $(1+\epsilon')$-approximate $(k+2)$-dimensional Steiner tree T' for the lifted terminal set P' with Aroras PTAS [2] that has a polynomial running in bounded dimension. Then we apply Lemma 3 to obtain a two-level Steiner tree $T = (T_{top}, T_1, \ldots, T_k)$ for $P_1, \ldots, P_k$ with length at most $l(T') - kK$. Let $T'^\star$ and $T^\star$ be optimum Steiner trees for P' and P. Since $l(T^\star) \geq K$ the length of T is

$$\begin{aligned} l(T) &\leq l(T') - kK \leq (1+\epsilon')l(T'^\star) - kK \\ &\leq (1+\epsilon')l(T^\star) + (1+\epsilon')kK - kK \\ &\leq (1+(k+1)\epsilon')l(T^\star) = (1+\epsilon)l(T^\star). \end{aligned}$$

□

4 Predetermined Connection Points

In all algorithms of this section we predetermine a connection point q_i for each set P_i $(i = 1, \ldots, k)$ and then call a Steiner tree approximation algorithm for $\{q_1, \ldots, q_k\}$ to get T_{top} and $P_i \cup \{q_i\}$ to get T_i $(i = 1 \ldots, k)$. We use the fact that we consider rectilinear instances to obtain better approximation factors than in Section 2.

4.1 Bounding Box Center

A natural approach is to choose each connection point as the center of the bounding box $B(P_i)$.

Theorem 5 *Using bounding box centers as connection points, we get a 1.75α-factor approximation algorithm for the two-level rectilinear Steiner tree problem, when using an α-factor approximation algorithm for rectilinear Steiner trees as a subroutine.*

A detailed analysis can be found in [8], which also provides a simple example attaining this factor.

4.2 Adjusted Bounding Box Center

We can improve the approximation factor by a more careful choice of the connection point. For a set P_i $(i \in \{1, \ldots, k\})$, we call the coordinate system with origin in the central point of its bounding box the *coordinate system of P_i*.

If a set P_i of terminals contains an element in each quadrant of its bounding box $B(P_i)$, we call the bounding box $B(P_i)$ *complete*. For subtrees with a complete bounding box we choose the connection point to the top-level tree as the central point of the bounding box as in Section 4.1 and compute a Steiner tree T_i' for $P_i \cup \{q_i\}$ as follows: We compute a Steiner tree T_i' for P_i. Thereby, we embed maximal paths in T_i' containing only Steiner vertices with degree two so that each such path has minimum distance to q_i while preserving its length. We then add an edge from q_i to a_i, where a_i is a point in T_i' minimizing the distance to q_i.

Let $T^\star = \{T^\star_{top}, T^\star_1, \ldots, T^\star_k\}$ be an minimum two-level Steiner tree. Again we choose $T^\star_{top}$ under all minimum two-level Steiner trees as large as possible so that there is a connection point $q_i^\star \in T^\star_{top} \cap T^\star_i \subseteq B(P_i)$.

We present the main ingredient for the analysis of the approximation ratio.

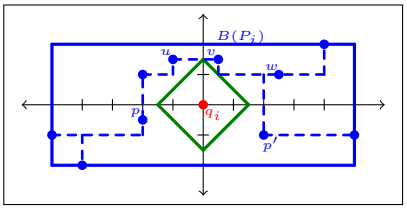


Figure 4: An example of the situation in the proof of Lemma 6. The green diamond is the l_1-circle with radius h around q_i.

Lemma 6 *Let $i \in \{1, \ldots, k\}$ and T_i', a_i be constructed as above. If $B(P_i)$ is complete, then $l(B(P_i)) + \|q_i - a_i\|_1 \leq l(T_i^\star)$.*

Proof. Define $h := \|q_i - a_i\|_1$. Since $B(P_i)$ is complete, T_i' intersects at least three of the four axes of the coordinate system to P_i. We assume w.l.o.g. that T_i' intersects the left, upper and right axis.

By the choice of T_i' and h there exist (see also Figure 4) for all $z \in [0, h]$

$$\begin{aligned} p &\in \{(x,y) \in P_i : x \leq -h, y \leq 0\}, \\ p' &\in \{(x,y) \in P_i : x \geq h, y \leq 0\}, \\ u &\in \{(x,y) \in P_i : x < 0, y \geq h\}, \\ v_z &\in V_z := \{(x,y) \in P_i : x \geq z, y \geq h - z\}. \end{aligned}$$

Let $v_h \in V_h$. If the unique $T_i^\star$-paths from p to u and from p' to v_h intersect, then $T_i^\star$ connects the lines $\{(x,y) : x = 0\}$ and $\{(x,y) : x = h\}$ twice, and therefore $l(B(P_i)) + h \leq l(T_i^\star)$.

Otherwise, we can choose a minimum $z \geq 0$ such that there is a $v \in V_z$ and the unique $T_i^\star$-paths from p to u and from p' to v are disjoint. The lines $\{(x,y) : y = 0\}$ and $\{(x,y) : y = h - z\}$ are connected twice in $T_i^\star$. Therefore we get $l(B(P_i)) + h - z \leq l(T_i^\star)$.

If $z = 0$ we are done. Otherwise, if our statement is false there is an $0 < \epsilon \leq z$ such that $\epsilon = l(B(P_i)) + h - l(T_i^\star)$ and a $w \in V_{z-\frac{\epsilon}{2}}$. Since the unique $T_i^\star$-paths from p to u and from p' to w are not disjoint, $T_i^\star$ connects the lines $\{(x,y) : x = 0\}$ and $\{(x,y) : x = z - \frac{\epsilon}{2}\}$ twice. Therefore we get the contradiction

$$\begin{aligned} \epsilon &= l(B(P_i)) + h - l(T_i^\star) \\ &\leq l(B(P_i)) + h - l(B(P_i)) - h + z - z + \tfrac{\epsilon}{2} = \tfrac{\epsilon}{2}. \end{aligned}$$

□

With Lemma 6 we tradeoff between the cost $\|q_i^\star - q_i\|_1 \leq l(B(P))$ to connect q_i to $T_{top}^\star$ and the cost $\|q_i - a_i\|_1$ to connect q_i to T_i'. From this we could derive an approximation factor of 13/8 if all partitions are complete. In general this is not the case and for incomplete bounding boxes we shift the connection points (and the coordinate system) towards the actual terminals as in Figure 5. A careful analysis using a similar version of Lemma 6 on the shifted coordinate system (details can be found in [8]) gives us the following result:

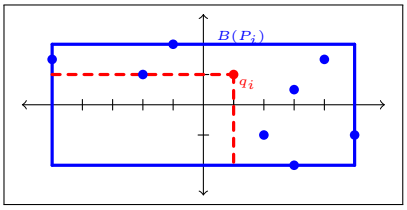


Figure 5: An example for an incomplete bounding box. The connection point q_i is shifted diagonally to the upper right until the red box hits a terminal.

Theorem 7 *There is a 2.37-factor approximation algorithm with runtime $\mathcal{O}(n \log n)$ and an 1.63-factor approximation algorithm in poly-time for the two-level rectilinear Steiner tree problem.*

References

[1] E. Alvarez-Miranda, I. Ljubic, S. Raghavan, and P. Toth: Recoverable Robust Two-Level Network Design Problem. *INFORMS Journal on Computing*, to appear, 2014.

[2] S. Arora: Polynomial time approximation schemes for Euclidian Traveling Salesman and other geometric problems. *Journal of the ACM*, 45(5): 753–782, 1998.

[3] M. Baou and F. Barahona. A polyhedral study of a two level facility location model . *RAIRO - Operations Research*, 48: 153–165, 2014.

[4] J. Byrka and B. Rybicki. Improved LP-Rounding Approximation Algorithm for k-level Uncapacitated Facility Location. *ICALP '12*, 157–169, 2012.

[5] A. Balakrishnan, T. L. Magnanti, and P. Mirchandani: A dual-based algorithm for multi-level network design. *Management Science*, 40(5): 567–581, 1994.

[6] J. R. Current, C. S. ReVelle, and J. L. Cohon: The hierarchical network design problem. *European Journal of Operational Research*, 27(1), 57–66, 1986.

[7] M. Hanan: On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14(2): 255–265, 1966.

[8] S. Held and N. Kämmerling: Two-Level Rectilinear Steiner Trees. `http://arxiv.org/abs/1501.00933`, 2015.

[9] S. Safra and O. Schwartz: On the complexity of approximating TSP with neighborhoods and related problems. *Computational Complexity*, 14(4): 281–307, 2006.

[10] T.L. Snyder: On the exact location of Steiner points in general dimension. *SIAM Journal on Computing*, 21(1): 163–180, 1991.

[11] H. Xiang, H. Ren, L. Trevillyan, L. Reddy, R. Puri, and M. Cho: Logical and physical restructuring of fan-in trees. *ISPD '10*, 67–74, 2010.

[12] H. Xiang, L. Reddy, L. Trevillyan, and R. Puri: Depth Controlled Symmetric Function Fanin Tree Restructure. *ICCAD '13*, 585–591, 2013.

Max Shortest Path for Imprecise Points

Yann Disser, Matúš Mihalák, Sandro Montanari

Abstract

We are given a graph $G = (V, E)$ with n vertices where each vertex corresponds to one of n given polygons in the plane and two vertices $s, t \in V$. We consider the problem of placing a point inside each polygon in order to maximize the shortest path distance between s and t in the resulting geometric graph. We show that the problem is hard to approximate for any factor $(1 - \epsilon)$ with $\epsilon < 1/4$, even when the polygons consist only of vertical aligned segments. For the special case where G is a path, we provide an algorithm computing an optimum placement in time $O(n \cdot k^2)$, where k is the maximum number of corners of a polygon.

1 Introduction

A *geometric graph* is a graph where each vertex corresponds to a point and the weight of an edge is the distance between the points it connects. We consider a set of polygons $\mathcal{P}$, an underlying graph connecting them, two polygons $P_s, P_t \in \mathcal{P}$, and the task of placing a point inside each polygon such that the shortest path distance between the points in P_s and P_t in the resulting geometric graph is maximum among all possible placements.

Imprecise measurements in real world applications require to design techniques that are able to cope with the uncertainty introduced by the unreliable estimations. A natural question in this setting concerns the computation of the minimum and maximum values that a geometric object can attain when the exact location of each point is unknown, but instead a region is known that contains it (an *imprecise point set*). For example, for imprecise points sets in the form of segments, squares, or disks, the problems of computing the smallest minimum spanning tree [2, 3, 8], the smallest bounding box [9], the smallest enclosing circle [5, 9], the farthest pair [9], the width [9], the closest pair [9], or the area and perimeter of the convex hull [8] have been studied.

Another problem that has been researched intensively in this setting is known as the Touring Polygons Problem (in short TPP). Given a sequence of simple polygons and two points s and t, the problem asks for a shortest tour starting at s and ending at t while visiting all polygons in the given order. The term "tour" comes from the fact that typically (but not necessarily) s and t are the same point. This problem is known to be solvable in polynomial time whenever the polygons are convex and disjoint [4]. It becomes NP-hard for any metric L_p, $p \geq 1$, as soon as we allow non-convexity both for disjoint [1] and overlapping [4] polygons. The only variant of the TPP with non-convex polygons admitting a polynomial time algorithm is known for the case of rectilinear axis-aligned polygons under the L_1 metric [4]. If we relax the requirement of computing an exact solution, approximation algorithms for the TPP in the non-convex case are known [6, 10].

The Shortest Path Problem for imprecise points (in short SPP) is a natural generalization of the TPP. Instead of an ordered sequence of polygons we are now given a (directed) graph connecting them; the presence of an edge indicates an "allowed traversal" from one polygon to the other. Given a starting and an ending polygon P_s and P_t, we search for a placement minimizing the shortest path distance of the points in P_s and P_t in the resulting geometric graph. In general, the SPP is NP-hard for any metric L_p, $p \geq 1$, even for disjoint segments whose angles with the x-axis are in $\{0, \pm\pi/4, \pi/2\}$. The proof is a straightforward generalization of the NP-hardness proof of the TPP in the case of non-convex disjoint polygons [1]. However, the SPP has been proven to be solvable in polynomial time under the L_1 metric if the polygons are (not necessarily convex) axis-aligned rectilinear [2].

So far, the SPP has been studied only in its minimum variant, where we look for placements minimizing the shortest path distance between the points in the given polygons. In this paper we consider the maximum variant, MAX-SPP, and look for placements maximizing the shortest path distance between s and t in the resulting geometric graph. We show that the problem is hard to approximate for any approximation factor $(1 - \epsilon)$ with $\epsilon < 1/4$, even when the polygons consist only of vertically aligned segments. We also consider the maximum variant of the TPP and propose an algorithm computing a maximum placement in $O(n \cdot k^2)$ time, where n is the number of polygons and k is the largest number of corners of a polygon in the sequence. The algorithm can be applied also in the case where the polygons are non-convex and overlapping, generalizing the only previously known result on MAX-TPP [7] that considers only the case of squares and line segments.

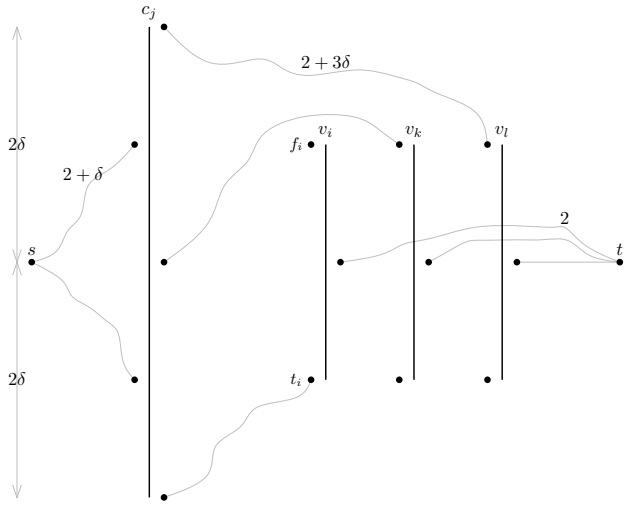


Figure 1: The gadget for $c_j = \overline{v_i} \vee v_k \vee v_l$

2 Max Shortest Path

We now show the hardness of approximation for MAX-SPP by providing a gap-producing reduction from 3-SAT. Given a 3-SAT instance, we construct a MAX-SPP instance and provide two threshold values τ_1, τ_2 with $\tau_1 - \tau_2 = 2\delta$ for an arbitrary constant $\delta > 0$. We show that the MAX-SPP instance admits an optimum solution with weight at least τ_1 if and only if the 3-SAT formula admits a satisfying assignment. If the formula does not admit a satisfying assignment, the weight of an optimum solution is at most τ_2.

In the construction of the MAX-SPP instance we will make use of *degenerate polygons*, i.e., points. Note that in any placement the weight of an edge connecting two degenerate polygons is fixed. We will make use of this observation in the construction of the MAX-SPP instance when assuming that the weight of an edge connecting two degenerate polygons p and q can be set to any weight greater or equal than $\|p-q\|_1$. This can easily be achieved by splitting the edge between p and q and introducing an intermediate point at a suitable distance from both.

The m clauses and n variables of the 3-SAT formula are represented in the MAX-SPP instance with gadgets sketched in Figure 1. There is a gadget for every clause, one for every variable, and two additional points s, t. The point s is located at the origin of the plane and t is located at coordinates $(3, 0)$.

The MAX-SPP instance contains $m + n$ vertically aligned segments. The middle points of these segments are equally spread along the x-axis, with the leftmost and rightmost ones respectively at coordinates $(1, 0)$ and $(2, 0)$. Given an arbitrary constant $\delta > 0$, the first m segments have length 4δ, and each of them corresponds to a clause of the 3-SAT formula. The remaining n segments have length 2δ, and each of them corresponds to a variable. We use v_i to denote the i-th variable and the segment associated with it; it will be clear from the context to which of the two we are referring to. Similarly, we use c_j to denote the j-th clause and the corresponding segment.

Variable gadgets. Given a segment v_i at x-coordinate x_i, we place a point to its right, at coordinates $(x_i + \frac{1}{4\gamma}, 0)$, and two points to its left, at coordinates $(x_i - \frac{1}{4\gamma}, -\delta), (x_i - \frac{1}{4\gamma}, \delta)$, where $\gamma := m+n$. We use t_i and f_i to denote respectively these last two points. We introduce in the underlying graph edges connecting v_i to each of these three points. Furthermore, the point to the right of v_i is also connected to t with an edge of weight 2. Points t_i and f_i are connected in a way specified in the following to gadgets of clauses where v_i appears as a literal. In the reduction, a point placed in the (open) bottom half of v_i corresponds to assign the value "true" to v_i. Conversely, a point in the (closed) top half of v_i corresponds to assign the value "false" to v_i.

Clause gadgets. Given a segment c_j at x-coordinate x_j, we place 5 points next to it, at coordinates $(x_j - \frac{1}{4\gamma}, -\delta), (x_j - \frac{1}{4\gamma}, \delta), (x_j + \frac{1}{4\gamma}, -2\delta), (x_j + \frac{1}{4\gamma}, 0), (x_j + \frac{1}{4\gamma}, 2\delta)$. We introduce in the underlying graph edges connecting c_j to each of these points. Among these points, those located to the left of c_j are also connected to s with edges with weight $2 + \delta$. The remaining points are connected to variable gadgets as follows. We associate the points at coordinates $(x_j + \frac{1}{4\gamma}, -2\delta), (x_j + \frac{1}{4\gamma}, 0), (x_j + \frac{1}{4\gamma}, 2\delta)$ respectively with the variables appearing in c_j as literals. If v_i appears in c_j as a positive literal, we connect the point associated to it to the point f_i of the variable gadget of v_i. If v_i appears in c_j as a negative literal, we connect the point associated with it to t_i. We set the weight of these edges to $2 + 3\delta$.

Theorem 1 *For any metric L_p, $p \geq 1$, it is* NP*-hard to approximate MAX-SPP for any factor $(1 - \epsilon)$ with $\epsilon < \frac{1}{4}$.*

Proof. We first create a MAX-SPP instance containing a gadget for every clause and every variable of the given 3-SAT. Then, we consider every path between the points s and t in the underlying graph of the constructed MAX-SPP instance and provide bounds on its weight in any placement. For simplicity, in the following we assume that distances are measured using the L_1 metric. It is however easy to see that similar bounds hold also for any metric L_p with $p \geq 1$.

For every variable v_i there is an st-path for every clause where v_i appears as a literal. By construction of the variable gadgets, we can assume without loss of generality that the point in segment v_i in any placement to be located either in one of its endpoints or in its middle point.

Suppose there exists a satisfying assignment for the 3-SAT instance. We then place a point arbitrarily in the top endpoint of a segment v_i if its value in the assignment is false and in the bottom otherwise. If a clause c_j is satisfied by v_i, we place in the segment c_j a point at the same y-coordinate as the point to the right of c_j associated with v_i. If c_j is satisfied by more than one variable, we choose one of them arbitrarily.

Consider now any st-path through a clause c_j and a variable v_i. If v_i is the chosen variable for c_j, the weight of the path between a point to the left of c_j to the point to the right of c_j associated with v_i is at least $\delta + \frac{1}{2\gamma}$. Without loss of generality, let v_i appear with positive sign in c_j. The st-path visits f_i and the point in v_i in the placement is located at its bottom. Thus, the weight of the path between f_i and the point to the right of v_i is at least $3\delta + \frac{1}{2\gamma}$. Since the remaining edges of the path have fixed weight, the overall weight of the st-path in the placement is

$$\tau_1 := 6 + 8\delta + \frac{1}{\gamma}.$$

If v_i is not the chosen variable for c_j, the weight of the path between the point to the left of v_i to its right might be smaller, but still at least $\delta + \frac{1}{2\gamma}$. On the other hand, the weight between the point to the left of c_j to its right is only greater, at least $3\delta + \frac{1}{2\gamma}$. Also in this case, the weight of the st-path is at least τ_1.

Suppose now that a satisfying assignment does not exist, and consider an optimum placement and the corresponding assignment of variables according to whether the point in segment v_i lies in its top or bottom half. Since the formula is not satisfiable, we can always find a non satisfied clause c_j; let v_i be a variable appearing in c_j. It is easy to see that there always exists an st-path visiting both c_j and v_i with weight at most

$$\tau_2 := 6 + 6\delta + \frac{1}{\gamma}.$$

Suppose that there exists a polynomial-time algorithm approximating MAX-SPP for some factor $(1-\epsilon)$ with $\epsilon < 1/4$. For any instance of 3-SAT, we then construct the above gadgets and calculate τ_1 and τ_2. Since $\tau_1 - \tau_2 = 2\delta$, setting δ such that $\frac{2\delta}{\tau_1} > \epsilon$ and using the approximation algorithm, we would be able to decide whether the weight of an optimum solution is greater or smaller respectively than τ_1 or τ_2. Thus, we could determine in polynomial time whether there exists a satisfying assignment for the 3-SAT instance. □

We observe that, in the above proof, the fact that there do not exist polynomial-time algorithms for MAX-SPP with approximation factor better than 3/4 comes from the ratio $\frac{2\delta}{\tau_1}$ being $\approx \frac{1}{4}(\frac{\delta}{\delta+1})$. It is an interesting open problem to determine whether there exist an algorithm with approximation factor exactly 3/4 (therefore making the above bound tight), or whether the problem cannot be approximated in polynomial time even for $\epsilon \geq 1/4$.

3 Max Touring Polygons

In the Max Touring Polygons Problem (or MAX-TPP for short), we are given an ordered sequence of polygonal regions $P_1, \ldots, P_n$ and two points s, t. We are asked for a longest tour that starts at s, visits the polygons in the given order, and ends at t. The MAX-TPP is a special case of MAX-SPP where the underlying graph connecting the polygons is a path.

Contrary to the general case, we will show that a solution to MAX-TPP can be found in polynomial time, even if the polygons are non-convex and overlapping. The algorithm for computing such a solution is based on the observation that the points belonging to an optimum placement form a subset of the corners of the n polygons. This observation is proved by the following lemma. In the following, we assume that distances are measured with the L_2 metric. It is straightforward to generalize the proofs for any metric L_p with $p \geq 1$.

Lemma 2 *There exists an optimum MAX-TPP placement such that every point in the placement is a corner of the corresponding polygon.*

Proof. We prove the statement by showing that, given a placement resulting in a longest s-t tour (an *optimum placement*) we can construct an equivalent placement $\boldsymbol{p'}$ where every $p_i \in \boldsymbol{p'}$ is a corner of the corresponding polygon P_i.

First, we show that there always exists an optimum placement where no point lies in the interior of the corresponding polygon. Consider a point $p_i \in \boldsymbol{p}$ lying in the interior of P_i and the points p_{i-1} and p_{i+1} (where $p_0 := s$ and $p_{n+1} := t$). If $p_{i-1} = p_i$ moving p_i away from its position in any direction would not decrease the weight of the optimum placement. If $p_{i-1} \neq p_i$, consider the ray from p_{i-1} towards p_i. This ray crosses a point q on the perimeter of P_i *after* crossing p_i. Furthermore, the triangle induced by p_{i-1}, p_i, p_{i+1} is completely contained in the triangle induced by p_{i-1}, q, p_{i+1}. Therefore, by triangle inequality the weight of the tour does not decrease if we replace p_i with q. We can repeatedly apply this statement for every point of $\boldsymbol{p}$ until every $p_i \in \boldsymbol{p}$ lies on the perimeter of the corresponding polygon P_i.

Given an optimum placement $\boldsymbol{p}$ where every point lies on the perimeter of the corresponding polygon, we show how to construct an optimum placement where every point lies on a corner. Consider a point $p_i \in \boldsymbol{p}$ not lying on a corner of P_i and the segment $\overline{ab}$ of the perimeter of P_i on which p_i lies. Since the sum $\|p_{i-1} - r\|_2 + \|r - p_{i+1}\|_2$ is convex for every $r \in$

$\overline{ab}$ the maximum value is obtained when r is one of the endpoints, either a or b; assume without loss of generality that it is maximized when r is equal to a. Therefore, the weight of the tour does not decrease if we replace p_i with a (that is a corner of P_i). By repeatedly applying this statement, we can replace every point of $\boldsymbol{p}$ not lying on a corner until every $p_i \in \boldsymbol{p}$ lies on a corner of the corresponding P_i. □

3.1 Algorithm

We can use the above theorem to design an algorithm computing an optimum MAX-TPP placement. Our solution uses an auxiliary graph $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$, in which there is a vertex $v \in V_\mathcal{D}$ for every corner of a polygon P_i and two special vertices $v_s, v_t \in V_\mathcal{D}$ corresponding to the points s, t. There is an edge $(u, v) \in E_\mathcal{D}$ for every vertex u corresponding to a corner of P_i and every vertex v corresponding to a corner of P_{i+1}, for $i = 1, \ldots, n-1$. Furthermore, there is an edge from v_s to every vertex corresponding to a corner of P_1 and from v_t to every vertex corresponding to a corner of P_n. We assign a weight to each edge equal to the distance between its endpoints. The following theorem shows that a longest path between v_s and v_t in $\mathcal{D}$ corresponds to an optimum MAX-TPP placement.

Theorem 3 *Given a longest v_s-v_t path $P = u_0, \ldots, u_{n+1}$ where $u_0 = v_s$ and $u_{n+1} = v_t$, the placement $\boldsymbol{p}$ where every $p_i \in \boldsymbol{p}$ corresponds to u_i for every $i = 1, \ldots, n$ is an optimum MAX-TPP placement.*

Proof. Observe that the weight of P is equal to the weight of the tour induced by $\boldsymbol{p}$. For the sake of contradiction, assume that there exists a placement $\boldsymbol{q}$ inducing a tour that is longer than the one induced by $\boldsymbol{p}$. By Theorem 2 we can assume the points of $\boldsymbol{q}$ to be corners of the polygons $P_1, \ldots, P_n$. Thus, the vertices corresponding to the points of $\boldsymbol{q}$ are connected in $\mathcal{D}$, and we can find a path Q from v_s to v_t following these vertices of weight larger than that of P. □

Computing a longest path in a graph is in general NP-hard. However, the graph $\mathcal{D}$ is not a general graph but a directed acyclic graph (DAG). To see this note that there are edges only from vertices of P_i to vertices of P_{i+1}. A longest path between v_s and v_t in a DAG can be found using a trivial dynamic programming algorithm with run-time $O(|V_\mathcal{D}| + |E_\mathcal{D}|)$. To evaluate $|V_\mathcal{D}|$ and $|E_\mathcal{D}|$, let k be the largest number of corners of a polygon $P_1, \ldots, P_n$. Since $\mathcal{D}$ contains a vertex for every corner of a polygon, one corresponding to s and one to t, we get $|V_\mathcal{D}| \geq n \cdot k + 2 \in O(n \cdot k)$. Furthermore, there is an edge from $v \in V_\mathcal{D}$ corresponding to a corner of P_i to every vertex corresponding to a corner of P_{i+1}, for every $i = 0, \ldots, n$ (where P_0 and P_{n+1} are degenerate polygons corresponding to the points s and t). Therefore, every vertex of $V_\mathcal{D}$ is adjacent to at most k edges, that is, $|E_\mathcal{D}| \in O(n \cdot k^2)$. The time to find an optimum MAX-TPP placement is thus $O(n \cdot k^2)$.

Acknowledgments

This work was supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS).

References

[1] A. Ahadi, A. Mozafari, and A. Zarei. Touring disjoint polygons problem is NP-hard. In *COCOA*, pages 351–360, 2013.

[2] Y. Disser, M. Mihalák, S. Montanari, and P. Widmayer. Rectilinear shortest path and rectilinear minimum spanning tree with neighborhoods. In *ISCO*, pages 208–220, 2014.

[3] R. Dorrigiv, R. Fraser, M. He, S. Kamali, A. Kawamura, A. López-Ortiz, and D. Seco. On minimum- and maximum-weight minimum spanning trees with neighborhoods. In *WAOA*, pages 93–106, 2012.

[4] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *STOC*, pages 473–482, 2003.

[5] S. Jadhav, A. Mukhopadhyay, and B. K. Bhattacharya. An optimal algorithm for the intersection radius of a set of convex polygons. *J. Algorithms*, 20(2):244–267, 1996.

[6] F. Li and R. Klette. Rubberband algorithms for solving various 2d or 3d shortest path problems. In *ICCTA*, pages 9–19, 2007.

[7] M. Löffler and M. Kreveld. Largest and smallest tours and convex hulls for imprecise points. In *SWAT*, pages 375–387, 2006.

[8] M. Löffler and M. Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

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

[10] X. Pan, F. Li, and R. Klette. Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons. In *CCCG*, pages 175–178, 2010.

Fully autonomous Self-Localization via Trajectory Representations based on Inflection Points

Stefan Funke [i] Robin Schirrmeister[ii] Simon Skilevic [ii] Sabine Storandt [ii]

Abstract

We present a novel method for self-localization in a road network in a fully autonomously manner, in particular not requiring GPS, GSM or WiFi availability. The new method does not rely on distance information but only absolute directional information that can be easily acquired using an electronic compass present in almost every current smartphone. Our approach is based on an inflection-point-based representation of trajectories in the road network and a respective query data structure.

1 Introduction

The central component of any navigation system is localizing oneself in the world. Common localization schemes nowadays are **GPS/GLONASS** (satellite-based with a precision up to a few meters), **GSM** (cellphone network based, precision 50m–1500m), and **Wifi** (based on georeferenced access point BSSIDs, precision 20m–100m). Except for GPS/GLONASS, all these methods require the interaction with third parties when acquiring one's own position.

A natural question to ask is, whether it is possible to localize oneself without interacting with third parties. Note that GPS/GLONASS is also not always available, e.g. due to obstructions by foliage or buildings. Our basic assumption is that we are moving in a road network and the central question is whether we can use information about how we move around together with knowledge of the underlying road network to find out where we are.

1.1 Related Work

In [7] we introduced the concept of *path shapes* which allows for fully-autonomous self-localization if one can measure both *distances* and *changes of direction*, e.g. from the onboard computer present in all modern cars. So we are given data of the form *"500m straight, then a 40 degrees left turn, 200m straight, then a 90 degrees right turn, ..."* and try to use this for self-localization in the road network which is known to us. For example, in Figure 1, the green path shape is given and we want to examine whether this shape appears somewhere in the road network – with rotations and imprecisions allowed. Our experiments on actual road networks showed that – when restricting to *quickest/shortest* paths on which individuals tend to move – a rather short prefix of a path typically suffices to make its shape unique (even under imprecisions). Based on that we constructed a query data structure based on a generalized suffix tree [9] which allows for very fast localization. Note that [7] essentially relies on a common notion of curve similarity (like Hausdorff or Frechet) and hence crucially depends on reliable distance information.

A very interesting similarity measure for curves was proposed by de Berg et al in [3]. They developed the so-called *direction-based Frechet-distance* which optimizes over all parameterizations for a pair of curves like the Frechet distance but differs from the latter in that it is based on differences between the directions of movement along the curves, rather than on positional differences hence being invariant under translations and scalings. The authors present algorithms to compute several variants of this direction-based Frechet distance in $O(nm)$ time. In retrospect, the intuition of our "directional path shapes" is very related to this measure proposed by de Berg and Cook. For our specific application scenario of self-localization, the running time of their algorithms as well as other algorithms for general curve matching problem (partial or complete) under Hausdorff or Frechet distance, e.g [6], [3], [2], [1] are prohibitive, though.

Our Contribution

In this paper we develop a new notion of 'path shape' together with a similarity measure related to the directional Frechet distance of [3]. Even with this seemingly less descriptive notion of path shape, characteristic representations of shortest paths exist with relatively short prefixes in real-world road networks. This allows for the construction of an efficient query data structure which supports fast self-localization queries. Compared to [7], the main novelty and advantage of our new path shape notion is the omission of distance measurements, so essentially having a compass available suffices for self-localization.

[i]Department of Computer and Information Science, Universität Stuttgart, Germany, funke@fmi.uni-stuttgart.de

[ii]Department of Computer Science, Universität Freiburg, Germany, skilevic@gmail.com, schirrmr,storandt@informatik.uni-freiburg.de

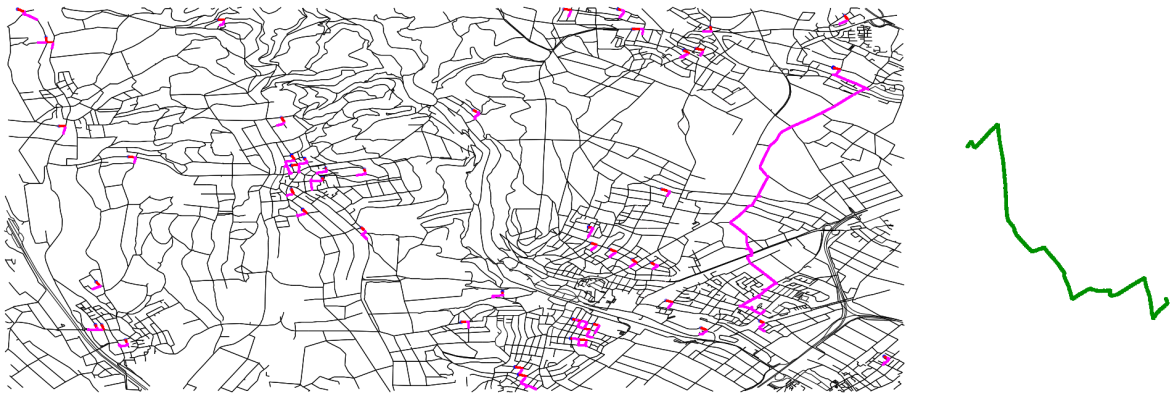


Figure 1: Previous result [7]: Path shape (green) and its partial matches (longer than 200m) in the Taunus area for a fuzzy comparison model. Blue dots indicate path origins. Only one full match exists determining the current position.

2 Directional Path Shapes (DPS) via Inflection Points

When travelling along a path π one can use an electronic compass to record the direction in which one is heading. When recorded continuously, this corresponds to a continuous function $\phi : [0, 1] \to C$ mapping each point along the path (parametrized over $[0, 1]$) to an angle of direction (represented as an angle/point on the unit circle C). So assume the same path π is travelled along by two vehicles at different speeds resulting in respective functions ϕ_1 and ϕ_2. Due to the different speeds aka parametrizations of the path, the two functions look different – in what respect could they be considered equal, though? One could imagine that the sequence of local minima or maxima of these functions is characteristic and the same for both functions ϕ_1 and ϕ_2. This is only makes sense, though, if we had mappings of the form $\phi : [0, 1] \to [0, 2\pi]$, that is to an *ordered* space, which is not really the case here since the space C is circular and has no notion of smaller or larger. A typical path includes both curves or turns to the left as well as curves or turns to the right, so a possible characterization is to consider the sequence of angles where there is a change from increasing angles to decreasing angles. In differential calculus this is called *inflection point* and characterized by the second derivative changing sign; in our case, the functions ϕ need not be differentiable, we will still call the respective angle inflection point (IP). It is clear that this sequence of inflection points is the same for ϕ_1 as for ϕ_2. If the 'boundary' between 0 and 2π is never crossed, these inflection points in fact correspond to local minima/maxima in the respective function which maps to $[0, 2\pi]$. See Figure 2 for an illustration of this concept.

So from now on we will represent a path by the ordered sequence of inflection points of ϕ which is invariant under different parametrizations of the respective path. This sequence of inflection points we call its *directional path shape (DPS)*.

3 Self-Localization via Matching Directional Path Shapes

Assuming we move on shortest (according to some known metric) in the network, the basic idea of our self-localization approach is as follows: In a preprocessing step we extract all shortest paths in the network, construct their directional path shapes (i.e. the sequence of inflection points), and store them in a suitable data structure. Now when driving around in the network we acquire a real-time DPS via reading out the built-in electronic compass of our smartphone. If there is a *unique* path shape in our database matching the just acquired DPS (having identical inflection points), we have determined our current location (up to the path segment between two inflection points). There are some challenges to make this idea practical, though:

- Acquiring a DPS real-time via a compass only rarely leads to exactly the same sequence of inflection points as the ground truth derived from the map; measuring errors or unsteady steering induce additional or differing inflection points.
- It is not feasible to compute *all* shortest paths even in a moderately sized network, let alone store their respective DPSs and query them.

3.1 Smoothing of Directional Path Shapes

Naturally, no sensor is flawless and the measured angles are perturbed by all kind of external factors. To take care of these fluctuations, we apply a line smoothing technique. For that purpose we convert the sequence of inflection points into a polyline by starting at $(0, 0)$ in a two-dimensional coordinate system, and then elongate the line by a straight segment in the direction of the angle corresponding to the first inflection point. We always use unit length for each straight segment. Then we apply the Douglas-Peucker algorithm [4] to this polyline which reduces the number

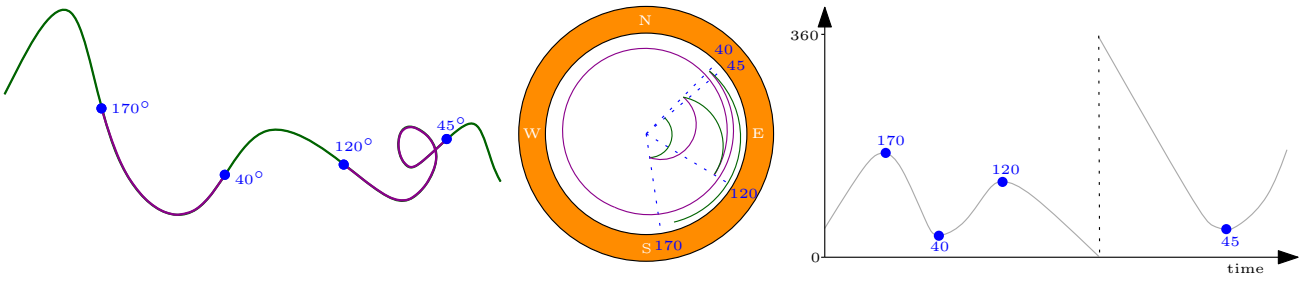


Figure 2: Left: Path with induced inflection points in blue; right turns in green, left turns in purple. Center and right: Two visualizations of the time-to-direction function ϕ. induced by traversing this path. On the right, inflection points correspond to local extrema.

of points but faithfully preserves the overall shape at the same time. Applying this smoothing to the DPS measured by the compass increases stability against measuring errors or unsteady steering.

3.2 A fault-tolerant Query Structure for DPSs

To accomplish fast localization queries, we follow the idea presented in [7] and transfer the problem of finding matching DPSs to a pattern search problem in texts. So the encoding of the current trajectory is regarded as a concatenated string over the alphabet $\{1, 2, \ldots, 360\}$. The text describing the network consists naively of all encodings of possible (shortest) paths in the map. A generalized suffix tree (GST) [9] requires only linear space and allows for efficient substring search by following a single path in the GST.

Unfortunately, constructing a GST for the DPSs of *all* $\Theta(n^2)$ shortest paths in the network is not feasible, hence we restrict to storing in the GST only the DPSs of all minimal shortest paths which have a unique DPS representation. Fault-tolerance at query time can be achieved by not only following a single path down the GST but all paths as long as the inflection point angles are within a certain error bound (in fact, in this case, we have to construct the GST for all minimal shortest paths with unique DPS representation even under fault tolerant comparisons).

4 Experimental Results

To show the practicability of our approach, we implemented the described algorithms and methods in C++ and tested them on several input networks. Timings were taken on a single core of an AMD Opteron 6172 with 2.1GHz Table 1 shows the characteristics of two of our benchmark networks, MA - Massachusetts and SG - Southern Germany, both extracted from OpenStreetMap data. We observe that the percentage of inflection points on an average shortest path is rather high, in fact about 50% (on the unsmoothed graph data).

Table 1: Characteristics of the used test graphs. Averages over 1000 random shortest paths (SPs).

	MA	SG
# nodes	294,345	5,588,146
# edges	731,874	11,711,088
avg. path length	120.4km	173.7km
avg. # IPs on SP	209	1373
avg. % IPs on SP	51.2	48.6

Table 2: Unique prefix length (in meters) dependent on the minimum number of inflection points (IPs) for exact queries, and with an angle tolerance of $t = 5$. Values are averaged over 1000 random shortest paths.

	MA	SG	MA	SG
	exact		$t = 5$	
0 IP	28,430	91,941	31,931	94,146
1 IP	21,388	45,083	31,293	81,934
2 IP	9,085	31,757	23,445	62,309
5 IP	114	1,596	8,853	4,998
10 IP	2	4	175	3

4.1 Characterizability of Street Networks

Our approach can only work if the inflection point representation is sufficiently differentiating paths in the network. So we conducted the following experiment: for a random shortest path π we searched for another path π' having the longest common prefix in its DPS representation with π. In Table 2 the average prefix lengths can be found, grouped by the minimum number of inflection points we demand the match to contain. If we set this number to zero, the longest match simply is the longest shortest path in the network with no encoding, completely independent of the reference path. Increasing the minimum number of inflection points, the prefix sizes decrease dramatically. Already using 5 IPs, the IPR becomes unique quite early, even when using an angle tolerance of $t = 5$ degrees between IPs. The first 10 IPs are rarely matched by any other path in the map, even for Southern Germany, this number is less than one percent of the total number of IPs on the path.

		MA	SG
exact	naive	17.1836s	245.6335s
	new	0.0022s	0.0332s
	speed-up	7810s	7398s
t=5	naive	19.0172s	261.1443s
	new	0.0038s	0.0458s
	speed-up	5004	5701

Table 3: Query times for self-localization averaged over 1000 random queries.

4.2 GST Construction and Querying

We constructed GSTs for all minimal paths with unique IPR, both under the exact as well as the imprecision-tolerant comparison model. For the road network of Southern Germany, the resulting GST can be employed to pinpoint one's location if the respective DSP has at least 17 inflection points in the exact model, 33 inflection points in case of the imprecision-tolerant model. Construction of the GSTs takes some time – about 2 days for the exact, and 4 days for the imprecision-tolerant model – and results in a data structure of about 4GB size which can still be stored on a mobile device.

With the GST we can perform self-localization queries several orders of magnitudes faster than the naive method which trys to realize a given DPS from every single node of the network, see Table 3 for the results. Even under a fault-tolerant comparison model and the large road network of Southern Germany, one can pinpoint one's own location in a fraction of a second.

4.3 Real-World Data Collection and Experiments

While the queries in Table 3 were randomly generated from the underlying road network, we also collected real compass data by bicycle with our Android compass app on a Nexus 4 over a period of a month. Cycling along our test route 20 times at quite different speeds, we could match 13 of the resulting DSPs correctly to the network using an angle tolerance of 5°. For the remaining ones, at least one inflection point differed more from the ground truth, the maximum was at 22°. These are preliminary results, the hardest challenge currently is still getting a reliable compass reading from the smartphone. The current Android API provides five different methods to get such a reading[1]: via a virtual orientation sensor (deprecated), magnetometer and accelerometer to rotation, low pass, rotation sensor, AHRS. Some of these methods require tuning parameters which we are still in the process of investigating. So there is a good chance that DPS can eventually be employed as a stand-alone method for precise self-localization once we can get sufficently good sensor readings.

[1] `http://developer.android.com/guide/topics/sensors/sensors_overview.html`

5 Conclusion

We have presented a novel way to characterize paths in a road network solely based on directional information. By examining real-world road-networks we could show experimentally that this characterization is expressive enough to distinguish shortest paths by a relatively short prefix. Based on this observation we developed an efficient data structure which quickly retrieves matching paths in a road network for a given path shape query and hence determines one's own location. Possible future work includes the consideration of more paths than just shortest/quickest. Our new method might also be interesting for other map matching applications [5, 8]. Like [7] our approach works better for European-style road networks, the many grid-type subnetworks e.g. in the U.S. lead to considerably longer path lengs to achieve unique IPRs.

References

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. Algorithms*, 49:262–283, 2003.

[2] E. M. Arkin, L. Chew, D. Huttenlocher, K. Kedem, and J. Mitchell. An efficiently computable metric for comparing polygonal shapes. In *1st Symp. on Discr. Algorithms(SODA)*, pages 129–137, 1990.

[3] M. de Berg and A. Cook IV. Go with the flow: The direction-based frechet distance of polygonal curves. In *Proc. 1st Int. ICST Conf. on Theory and Practice of Algorithms in Computer Systems (TAPAS)*, 2011.

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

[5] J. Eisner, S. Funke, A. Herbst, A. Spillner, and S. Storandt. Algorithms for matching and predicting trajectories. In *Proc. of the 13th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2011.

[6] M. Frenkel and R. Basri. Curve matching using the fast marching method. *EMMCVPR*, pages 35–51, 2003.

[7] S. Funke and S. Storandt. Path shapes an alternative method for map matching and fully autonomous self-localization. In *In 19th International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS)*, 2011.

[8] M. Quddus, W. Ochieng, and R. Noland. Current map-matching algorithms for transport applications: state-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15:312 – 328, 2007.

[9] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995. 10.1007/BF01206331.

A Method for Fitting Real World Tessellations with Voronoi Diagrams

Supanut Chaidee* Kokichi Sugihara*

Abstract

There are many phenomena that generate polygonal tessellations on surfaces in 3D space. In this study, we propose a framework for finding the spherical Voronoi diagram that best fits a given photograph of a tessellation on a curved surface. A new distance is introduced in order to define the Voronoi diagram that is suitable for this purpose. Finally, this is reduced to an optimization problem, and numerical results are shown.

1 Introduction

In the real world, there are many phenomena that generate polygonal tessellations, for example, the patterns made by animal territories or the surface of fruits. We are interested in checking whether a given tessellation belongs to a Voronoi diagram (or whether it is a Dirichlet tessellation).

Several criteria have been proposed for evaluating the difference between a particular tessellation and a Voronoi diagram. With a planar polygonal tessellation, Honda [2, 3] gave the framework for approximating the Voronoi generators from a given tessellation and defined the deviation between the given tessellation and the Voronoi diagram. In [6], the deviation between two tessellations, said the discrepancy, is defined in a different way from [2, 3], and the problem is reduced to the optimization problem for adjusting positions of the Voronoi generators to obtain the smallest discrepancy. Also, The tessellation fitting by considering the equation system was introduced in [1]. Khiripet, et. al. [4] also introduced the tools for analyzing the Voronoi diagram from a given photo. However, they are all for a 2D Voronoi diagram, whereas we are interested in tessellations on curved surfaces in 3D space.

In addition, many tessellations in the real world appear in 3D space. Some criteria are also studied in 3D space, for example, the method for fitting the 3D Laguerre Voronoi tessellation to foam structure in [5]. Fitting a given tessellation to the Voronoi diagram and evaluating the difference with respect to the generators of the given tessellation will be useful for constructing mathematical models of polygonal patterns found in nature.

*Graduate School of Advanced Mathematical Sciences, Meiji University, Japan, {schaidee, kokichis}@meiji.ac.jp

One example of a polygonal tessellation found in nature is jackfruit (*Artocarpus heterophyllus*), a multiple fruit that is found in various tropical locations. Photos of a jackfruit are shown in Figure 1. The surface of a jackfruit is covered by a complicated 3D microstructure, which, when viewed along a line perpendicular to the surface, forms a polygon-like tessellation in which each cell contains a blunt spike. Each tip of each spike is located on a flower of an inflorescence. Therefore, it could be said that the tip of each spike is the generator of polygonal tessellation.

Figure 1: (left) A jackfruit; (right) the skin of a jackfruit

In this study, we will assume that the pattern seen in the skin of a jackfruit can be approximated by a Voronoi diagram on a sphere. We provide a framework for using a photograph of a jackfruit to evaluate the difference between this pattern and a spherical Voronoi diagram. Because it is not appropriate to compare such a photo with a Voronoi diagram on a plane, we introduce a Voronoi diagram that reflects 3D structures on a plane. We then formulate an optimization problem to determine the best sizes for the sphere and the spike heights. Finally, we confirm the validity of our method by numerical results, using artificial and real tessellation patterns.

2 Problem Formulation

Our goal is to compare a given tessellation on a curved surface (including the centers of the tessellation cells) with the Voronoi diagram with respect to those centers on that surface. In particular, in this study, we will consider the pattern made by the skin of a jackfruit.

2.1 Modeling assumptions

For simplicity, we assume that locally, a jackfruit can be considered to be a sphere of radius R. The exact shape of a jackfruit is only approximately ellipsoidal, but we consider a relatively small subregion of the surface, in which this approximation is adequate. We also assume that the jackfruit has spikes of a uniform height h, that is, the tips of the spikes (the spike-dots) are on a sphere of radius $R + h$.

We also assume that the photo of the jackfruit is an orthogonal projection of the sphere onto the plane of the photo (the XZ-plane). The center of the photo is assumed to be at the center of the camera lens and the origin of the XZ-plane.

Further, we consider the spike-dots to be the generators of the Voronoi diagram on the sphere. We define the spike-dots rigorously, as follows.

The **spike-dot** s_i is the dot located at the tip of spike q_i. The set of spike-dots is denoted as $\mathcal{B} = \{s_1, ..., s_n\}$, where n is the number of spike-dots shown in the photo.

Starting from the photo, we would like to generate a tessellation that can be considered to be the projection of the pattern of the jackfruit skin onto the XZ-plane. The tessellation T that is extracted from a photo of a jackfruit is called a **jackfruit tessellation** (hereinafter, for simplicity, tessellation). We assume that T is a convex embedding of a planar 3-regular straight graph containing the set of spike-dots $\mathcal{B} = \{s_1, ..., s_n\}$.

From consideration of the physical reality, there exists a single spike-dot in each tessellation cell, but some spike-dots are not shown in the photo, because they are outside the frame.

2.2 Comparison of the difference in the two dimensions

Let C be a cell in a tessellation T. C is said to be a spike-dot cell if C contains a spike-dot. Otherwise, we call it a non-spike-dot cell. In order to compare two tessellations fairly, we construct tessellations T_g and V_g such that each tessellation cell of T_g and V_g is completely surrounded by spike-dot cells in the tessellations T and V, respectively.

Now, we have two tessellations with the same spike-dots. For each pair of cells corresponding to the same spike-dot, the intersection (overlapping area) is a convex polygon containing the spike-dot. The difference in area between tessellation T_g and the sum of the overlapping areas is denoted D_T. We denote by D_V the difference in area between the Voronoi diagram V and the sum of overlapping areas. Let A_T and A_V be the areas of tessellation T_g and the Voronoi diagram V_g, respectively. The difference between these two tessellations, denoted $\Delta_{T,V}$, can be determined as the ratio of the difference area to the overall area:

$$\Delta_{T,V} = \left(\frac{D_T + D_V}{2}\right) / \left(\frac{A_T + A_V}{2}\right) = \frac{D_T + D_V}{A_T + A_V}. \quad (1)$$

We note that D_T, D_V and A_T, A_V are not much different.

The value $\Delta_{T,V}$ indicates the extent of the difference between the jackfruit skin pattern and the Voronoi diagram. This ratio is called *discrepancy* hereafter.

2.3 Framework for projecting a Voronoi diagram on a sphere onto a plane

In order to evaluate the difference between the jackfruit tessellation and the Voronoi diagram, we must first construct the Voronoi diagram on the sphere. However, we do not know the size of the sphere.

We initially assume values R and h for the radius and the spike height, respectively. We project each spike-dot $s_i = (x_i, z_i)$ on the photo plane onto the sphere. Note that each spike-dot is derived from the top of a jackfruit spike, which is located on the top of a sphere of radius $R + h$. Hence, the coordinates of the tip of spike are $s_i(R + h) = (x_i, \sqrt{(R + h)^2 - (x_i^2 + z_i^2)}, z_i)$. We project this point onto a sphere of radius R, in which the center of the projection is at the center of the sphere. Thus, the Voronoi generators are obtained as $s_i(R) = (x_i', y_i', z_i')$, which are located on the sphere of radius R. Finally, we construct the Voronoi diagram with respect to the geodesic distance, V_S, on this sphere.

We project the vertices of V_S onto the XZ-plane. We approximate the Voronoi edges projected to a plane by straight line segments to obtain a convex tessellation V.

With the tessellation T extracted from the photo and the projected Voronoi diagram V, we obtain T_g and V_g, which consist of spike-dot cells. We are now ready to compare the two tessellations.

3 A New Distance Corresponding to the Voronoi Diagram on the Sphere with Spikes

We have already provided the framework for comparing a given tessellation that is extracted from the photo, and the Voronoi diagram on a sphere for the spike-dots. Our process begins with the coordinates on the photo plane (the XZ-plane). In this section, we define a new distance on the plane that corresponds to the distance on the sphere with spikes.

Let $P'(x_1, z_1)$ be a spike point, and let $Q'(x_2, z_2)$ be any point on the photo plane. Assume that P' comes from spike point P on the sphere with radius $R + h$, and Q' comes from general point Q on the sphere with radius R.

Based on these assumptions, we are able to specify the y-coordinates of the spike points on the sphere: $y_1 = \sqrt{(R+h)^2 - (x_1^2 + z_1^2)}$ and $y_2 = \sqrt{R^2 - (x_2^2 + z_2^2)}$. Therefore, P and Q are represented as $P = (x_1, \sqrt{(R+h)^2 - (x_1^2 + z_1^2)}, z_1)$ and $Q = (x_2, \sqrt{R^2 - (x_2^2 + z_2^2)}, z_2)$. Let P'' be the point of intersection of the line OP and the sphere with radius R, where O is the origin of the sphere. We can obtain the distance on the sphere by considering P and Q as vectors. Note that the circular arc length from P'' to Q is $d_{\text{gc}} = R \cdot \theta$, where R is the sphere radius, and θ is the angle between P and Q, measured at the center of the plane.

In particular, $\theta = \arccos\left(\frac{\mathbf{P}\cdot\mathbf{Q}}{|\mathbf{P}||\mathbf{Q}|}\right)$, where $|\mathbf{P}| = R + h$ and $|\mathbf{Q}| = R$.

Therefore, the distance between P' and Q' on the plane is

$$d_{\text{plane}}(P', Q') = R \cdot \arccos\left(\frac{\mathbf{P}\cdot\mathbf{Q}}{R(R+h)}\right). \quad (2)$$

This is the distance between a Voronoi generator corresponding to a spike-dot and an arbitrary point on the plane. Note that this distance does not satisfy the metric axiom, but we use the term 'distance' or 'metric' for convenience. With this distance, we are able to draw the Voronoi diagram on the XZ-plane.

From the distance defined by (2) and the framework presented in the previous sections, the following theorem is obtained directly.

Theorem 1 *The Voronoi diagram on the sphere with radius R generated by spikes on the sphere with radius $R + h$ is equivalent to the Voronoi diagram on the plane with respect to the distance defined by (2).*

4 Stability of Tessellation Fitting

If the Voronoi diagram on a plane with respect to the distance (2) is given, we can define the discrepancy. However, we do not know R, h, or the center of the sphere. Therefore, the discrepancy is a function of these parameters, and optimization is required to find the best-fit values.

4.1 Discrepancy function

For a given tessellation T, we generate a Voronoi diagram on the plane V with respect to the distance (2). With these two tessellations, we can compute the discrepancy defined in Section 2.3. We observe that the variables R, h, and the center of the sphere affect the discrepancy.

We define D as a discrepancy function that depends on the center of the sphere projected onto the plane (x, z), the sphere radius R, and the spike height h, which is the ratio $\Delta_{T,V}$ shown in the equation (1); that is, the discrepancy function is denoted by $D(x, z, R, h)$. In order to fit a given tessellation T with a Voronoi diagram V, optimization is used to find the appropriate x, z, R, h that minimizes $D(x, z, R, h)$. However, it is complicated to express the discrepancy function D explicitly.

In order to fit a given tessellation with the Voronoi diagram on the sphere, the framework is written as the following algorithm.

Algorithm: Spherical Voronoi Fitting
Input: $\mathcal{B}$ - set of spike dots on the xz-plane, T - jackfruit tessellation corresponding to $\mathcal{B}$, initial $\mathbf{x} = (x, z, R, h)$
Output: x, z, R, h which minimize $D(x, z, R, h)$
Definition: function $D(x, z, R, h)$
  Fix T as a given tessellation;
  for $i = 1$ **to** $|\mathcal{B}|$;
    Project s_i to $s_i(R + h)$;
    Project $s_i(R + h)$ to $s_i(R)$;
  end for;
  Construct V_S with respect to $\cup\{s_i(R)\}$;
  Project V_S to V;
  Intersect T and V cell by cell;
  Compute $D_T, D_V, A_T, A_V, \Delta_{T,V}$;
  $D(x, z, R, h) \leftarrow \Delta_{T,V}$;
end Definition
Procedure:
1. Calculate $D(\mathbf{x})$;
2. Search for $\mathbf{x}' = (x', z', R', h')$ such that $D(\mathbf{x}) > D(\mathbf{x}')$ (by the method described later);
3. **if** such $\mathbf{x}'$ is found **then**
   $\mathbf{x} \leftarrow \mathbf{x}'$; $D(\mathbf{x}) \leftarrow D(\mathbf{x}')$ and go to step 2;
   else
   report $\mathbf{x}$ and $D(\mathbf{x})$;
   end if;

end Procedure

4.2 Experiments and numerical results

We performed the experiments using an artificially generated Voronoi diagram with respect to known parameters, with $R = 18$ and $h = 0.4$, and where the center of the sphere was at $(-0.2, -0.2)$. In our experiments, this Voronoi tessellation will be assumed as a given jackfruit tessellation. We then generated a new Voronoi diagram V, where the center of the sphere was at $(-0.2, -0.2)$, and the parameters R and h varied. Now, the optimization problem can be considered as two independent variables, R and h. Note that the minimum of D is equal to 0 at the point $(18, 0.4)$.

In the minimization of D, observations of the behavior of the discrepancy function led us to use a circular search, which is the search for the location of (R, h) minimizing $D(x, z, R, h)$ when x, z are fixed. We began a circular search at the initial point (R_0, h_0) with discrepancy M_i. We then computed the discrepancies of 36 points on the circle centered at (R_0, h_0)

and with radius α and lying in the Rh-plane. These points were arranged counterclockwise in a sequence, and denoted $\mathcal{C} = \{p_1, p_2, ..., p_{36}\}$. Among the discrepancies of these 36 points, we chose the subsequence $\mathcal{C}_k = \{p_j, p_{j+1}, p_{j+2}\}$ of $\mathcal{C}$ satisfying the property that $D(p_j) \geq D(p_{j+1})$ and $D(p_{j+1}) \leq D(p_{j+2})$. For each subsequence, we used a binary search to divide the arc between p_j and p_{j+2} into five subdivisions. We then reevaluated the subsequences and conducted another binary search to obtain the minimum among the subsequence $\mathcal{C}_k$. We then found the minimum discrepancy among all subsequences of $\mathcal{C}$, denoted M_{i+1}. If $M_i \leq M_{i+1}$, we decreased α to $\alpha/2$ and reiterated the previous steps. Otherwise, we moved to that point and continued the search. The search was stopped when the radius had been decreased by a certain fixed factor.

This numerical experiment was implemented on Wolfram Mathematica®10. The initial values were chosen from 8 points in the different direction around the correct answer. Starting from all the initial values $(16.0, 0.3)$, $(16.0, 0.4)$, $(16.0, 0.8)$, $(18.0, 0.3)$, $(18.0, 0.5)$, $(20.0, 0.3)$, $(20.0, 0.4)$, $(20.0, 0.5)$, R and h converge to the answer $R = 18.00$ and $h = 0.4000$ averagely, where the standard deviation of R and h are 5.734×10^{-3} and 4.684×10^{-7}, respectively. The plot of the discrepancy values of some initial value is shown in the Figure 2.

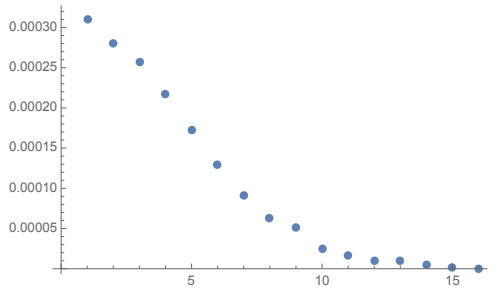


Figure 2: The change of the discrepancy function values from the initial value $(16, 0.3)$

From the results of experiments with artificially given tessellations, we conclude that we can acquire the corrected parameters from various initial values by using a circular search strategy.

5 Concluding Remarks

We proposed a framework for finding a spherical Voronoi diagram that can be fitted to the tessellation acquired from the photo of a tessellation on an approximately spherical surface covered by spikes. We also introduced a new distance for a Voronoi diagram on a sphere with spikes. Finally, we provided a method for optimizing the parameters in order to obtain the best-fit Voronoi diagram.

We can use the proposed method to verify that a given surface with generators can be considered to be a Voronoi diagram. However, our framework can be applied for fitting the polygon-like tessellation on the surface which is a convex surface covered by almost uniformly spikes. In practical, we have already done the experiments with the real data from a taken photo by using the proposed framework. The overall idea with the real data is shown in the Figure 3. Our framework will be useful for constructing mathematical models of the formation of polygonal patterns in various natural phenomena.

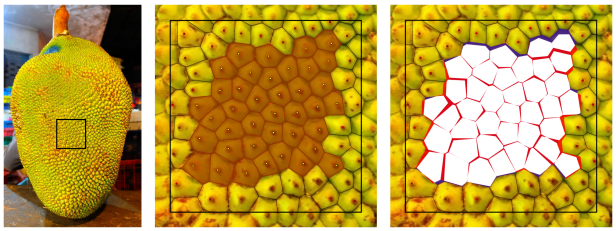


Figure 3: (left) A jackfruit photo with selected area; (middle) the extracted tessellation; (right) the difference of the given tessellation and the best-fit Voronoi diagram

Acknowledgments

The first author acknowledges the MIMS Ph.D. Program of the Meiji Institute for Advanced Study of Mathematical Sciences, Meiji University. This research is partly supported by the Grant-in-Aid for Basic Research No. 24360039 of MEXT.

References

[1] D. G. Evans and S. M. Jones *Detecting Voronoi (Area-of-Influence) Polygons* Mathematical Geology, 19: 6, 1987, 523-537.

[2] H. Honda *Description of Cellular Patterns by Dirichlet Domains: The Two-Dimensional Case* Journal of Theoretical Biology, 72: 3, 1978, 523-543.

[3] ———— *Geometrical Models for Cells in Tissues* International Review of Cytology, 81, 1983, 191-246.

[4] N. Khiripet, W. Khantuwan, and J. R. Jungck *Kame: a Voronoi Image Analyzer* Bioinformatics, 28:13, 2012, 1802-1804.

[5] C. Lautensack *Fitting three-dimensional Laguerre Tessellations to Foam Structure* Journal of Applied Statistics, 35: 9, 2008, 985-995.

[6] A. Suzuki and M. Iri *Approximation of a Tessellation of the Plane by a Voronoi Diagram* Journal of the Operation Research Society of Japan, 29: 1, 1986, 69-97.

Exact Minkowski Sums of Polygons With Holes

Alon Baram* Efi Fogel* Dan Halperin* Michael Hemmer† Sebastian Morr†

Abstract

We present an efficient algorithm that computes the Minkowski sum of two polygons, which may have holes. The new algorithm is based on the convolution approach. Its efficiency stems in part from a property for Minkowski sums of polygons with holes, which we prove to hold in any dimension. Given two polygons with holes, we can fill in all the holes of one polygon, transforming it into a simple polygon, and still obtain exactly the same Minkowski sum. Obliterating holes in the input summands speeds up the computation of Minkowski sums.

We introduce a robust implementation of the new algorithm, which follows the Exact Geometric Computation paradigm and thus guarantees exact results. We also present an empirical comparison of the performance of Minkowski sum constructions, where we show that the implementation of the new algorithm exhibits better performance than several other implementations in many cases. In particular, we compared the implementation of the new algorithm, an implementation of the standard *convolution* algorithm, and an implementation of the *decomposition* approach using various convex decomposition methods.

The software has been developed as an extension of the *2D Minkowski Sums* package of CGAL (Computational Geometry Algorithms Library). Additional information is available at `http://acg.cs.tau.ac.il/projects/rc`.

1 Introduction

The Minkowski sum of two sets P and Q is defined as $P \oplus Q = \{p + q \,|\, p \in P, q \in Q\}$. In this paper we focus on the computation of Minkowski sums of general polygons in the plane, that is, polygons that may have holes. However, some of our results also apply to higher dimensions. Minkowski sums are ubiquitous in many fields, including robot motion planning, assembly planning, and computer aided design.

*School of Computer Science, Tel Aviv University, Israel. {alontbst, efifogel}@gmail.com, danha@post.tau.ac.il. Work by E.F. and D.H. has been supported in part by the Israel Science Foundation (grant no. 1102/11), and by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

†Dept. of Computer Science, TU Braunschweig, Germany. mhsaar@gmail.com, sebastian@morr.cc. Work by S.M. and M.H. has been support by Google Summer of Code 2014.

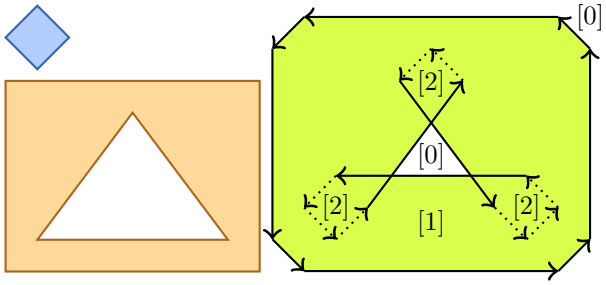


Figure 1: The convolution of a convex and a non-convex polygon; winding numbers are indicated in brackets; dotted edges are left out for the reduced convolution.

1.1 Terms, Definition, and Related Work

During the last four decades many algorithms were introduced to compute the Minkowski sum of polygons or polyhedra. For exact two-dimensional solutions see, e.g., [4]. For approximate solutions see, e.g., [7] and [9]. For exact and approximate three-dimensional solutions see, e.g., [6], [10], [11], and [15].

Computing the Minkowski sum of two convex polygons is rather easy using an operation similar to merging two sorted lists of numbers, as $P \oplus Q$ is a convex polygon bounded by copies of the edges of P and Q ordered according to their slope. If the polygons are not convex, it is possible to use one of the two following general approaches:

Decomposition Algorithms that follow the decomposition approach decompose P and Q into two sets of convex sub-polygons. Then, they compute the pairwise sums using the simple procedure described above. Finally, they compute the union of the pairwise sums. This approach was first proposed by Lozano-Pérez [12]. The performance of this approach heavily depends on the method that computes the convex decomposition of the input polygons. Flato et al. [1] described an implementation of the first exact and robust version of the decomposition approach, which handles degeneracies. They also tried different decomposition methods, but none of them though handled polygon with holes.

Convolution Let $V_P = (p_0, \ldots, p_{m-1})$ and $V_Q = (q_0, \ldots, q_{n-1})$ denote the vertices of the input polygons P and Q, respectively. Assume that their boundaries wind in a counterclockwise order around

their interiors. The *convolution* of these two polygons, denoted $P * Q$, is a collection of line segments of the form[1] $[p_i + q_j, p_{i+1} + q_j]$, when the vector $\overrightarrow{p_i p_{i+1}}$ lies counterclockwise in between $\overrightarrow{q_{j-1} q_j}$ and $\overrightarrow{q_j q_{j+1}}$ and, symmetrically, of segments of the form $[p_i + q_j, p_i + q_{j+1}]$, when the vector $\overrightarrow{q_j q_{j+1}}$ lies counterclockwise in between $\overrightarrow{p_{i-1} p_i}$ and $\overrightarrow{p_i p_{i+1}}$.

According to the *Convolution Theorem* stated in 1983 by Guibas et al. [5], the convolution is a superset of the Minkowski sum boundary. The segments of the convolution form a number of closed (possibly self-intersecting) polygonal curves called *convolution cycles*. On this basis, Wein [16] implemented the standard convolution algorithms for simple polygons. He computed the winding number[2] for each face in the arrangement induced by the convolution cycles and used it to determine whether the face as part of the Minkowski sum or not; see Figure 1. Wein's *exact* implementation is available in CGAL [14].

Kaul et al. [8] observed that a segment $[p_i + q_j, p_{i+1} + q_j]$ (resp. $[p_i + q_j, p_i + q_{j+1}]$) cannot possibly contribute to the boundary of the Minkowski sum if q_j (resp. p_j) is a reflex vertex (see dotted edges in Figure 1). The remaining subset of convolution segments, the *reduced convolution*, is still a superset of the Minkowski sum boundary but the idea of winding numbers can not be applied as there are no closed cycles anymore. Instead, Behar and Lien [2], first identify faces in the arrangement of the reduced convolution that may represent holes (based on proper orientation of all boundary edges of the face). Thereafter, they check whether such a face is indeed a proper hole by selecting a point x inside the face and performing a collision detection of P and $x \oplus -Q$. Their implementation exhibits faster running time than Wein's method, but may not be able to handle some degenerate cases: although they are using advanced multi-precision arithmetic, the detection of some degeneracies requires an exact approach. The method was also extended to three dimensions [11].

Milenkovic and Sacks [13] defined the *Monotonic Convolution*, which is another superset of the Minkowski sum boundary. They show that this set defines cycles and induces winding numbers, which are positive only in the interior of the Minkowski sum.

1.2 Our Results

We present an efficient algorithm that computes the Minkowski sum of two polygons, which may have holes. The new algorithm is a variant of the algorithm proposed by Behar and Lien [2], which computes the reduced convolution set. In our new algorithm, the initial set of filters proposed in [2] is enhanced by the removal of complete holes in the input, which reduces the size of the reduced convolution set even further. This enhancement is backed up by a simple theorem, the proof of which is also presented; see Section 2. Moreover, we show that at least one of the input polygons can always be made simple (before applying the convolution). This latter result is applicable to any dimension and independent of the used approach.

We introduce an implementation of the new algorithm. We also introduce implementations of two new convex decomposition methods that handle polygons with holes as input—one is based on vertical decomposition and the other is based on triangulation. These two methods can be directly applied to compute the Minkowski sum of polygons with holes via decomposition. All our implementations are robust and handle degenerated cases.

In addition, we present an empirical comparison of all the implementations above and existing implementations; see Section 4. We show that the implementation of our new algorithm exhibits better performance than all other implementations in many cases.

The software has been developed as part of the *2D Minkowski Sums* package of the Computational Geometry Algorithms Library (CGAL) [14], and as such, it is written in C++ and rigorously adheres to the generic-programming paradigm and the EGC paradigm.

2 Filtering Out Holes

The fundamental observation of the convolution theorem is that only points on the boundary of P and Q can contribute to the boundary of $P \oplus Q$. Specifically, the set of segments in the convolution $P * Q$ is a superset of the segments of the boundary of $P \oplus Q$.

The idea behind the reduced convolution method is to filter out segments of $P * Q$ that can not contribute to the boundary of $P \oplus Q$ using a local criterion; see Section 1.1. In this section we introduce a global criterion. The following theorem shows that we can ignore all segments in $P * Q$ that are induced by a hole if it is relatively small compared to the other polygon, meaning that the hole is completely irrelevant for $P \oplus Q$. In fact, it implies that the hole can be removed (that is, filled up) before the main computation starts, independently of the used approach; see also Figure 2 for an intuition.

Theorem 1 *Let H be the interior of a hole in polygon P. If there is a path γ contained in polygon Q that does not fit under any translation in $-H$, then H is irrelevant for the computation of $P \oplus Q$.*

Proof. Consider a point $p \in \partial H \subset P$ and a point $q \in \partial Q$ and assume for contradiction that $p \oplus q$ is on

[1] Addition of vertex indices is carried out modulo n (respectively m) for P (respectively Q).

[2] Informally speaking, the winding number of a point $p \in \mathbb{R}^2$ with respect to some planar curve γ is an integer number counting how many times does γ wind in a counterclockwise orientation around p.

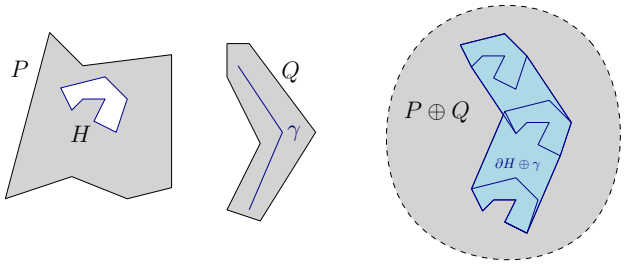


Figure 2: A small hole H is irrelevant for the computation of $P \oplus Q$ as adding ∂H and $\gamma \subset Q$ fills up any potential hole in $P \oplus Q$ related to H.

the boundary of $P \oplus Q$. First, observe that $p \oplus q$ must be part of the boundary of a hole $\overline{H}$ of $P \oplus Q$ as $\partial H \oplus q$ forms a closed cycle. For any point $x \in \overline{H} \not\subset P \oplus Q$ it must hold that $x \in H \oplus y \; \forall y \in Q$. Specifically, it must hold $\forall y \in \gamma \subset Q$. This is equivalent to $y \in (x \oplus -H)$ for all $y \in \gamma$, stating that γ fits into $-H$ under some translation, a contradiction. □

Corollary 2 *If the closed axis-aligned bounding box B_Q of Q does not fit under any translation in the open axis-aligned bounding box B_H of a hole H in P, then H does not induce a hole in $P \oplus Q$.*

Proof. W. l. o. g. assume that B_Q does not fit into B_H with respect to the x-direction. Consider the two extreme points of Q in that direction and connect them by a path γ, which obviously does not fit into $-H$, as it does not fit into B_H. □

Note that if B_Q does not fit in the open axis-aligned bounding box B_P of P, it cannot fit in the bounding box of any hole of Q, implying that all holes of Q can be ignored. Since for any two bounding boxes either $B_Q \not\subset B_P$ or $B_P \not\subset B_Q$ holds, we need to consider the holes of at most one polygon.

Consequently, we can remove all holes whose bounding boxes are, in x- or y-direction, smaller than, or as large as, the bounding box of the other polygon, as an initial phase of all methods. With fewer holes, convex decomposition results in fewer pieces. Moreover, when all holes of a polygon become irrelevant, one can choose a decomposition method that handles only simple polygons instead of a decomposition method that handles polygons with holes, which is typically more costly. As for the convolution approach, the intermediate arrangements become smaller, speeding up the algorithm.

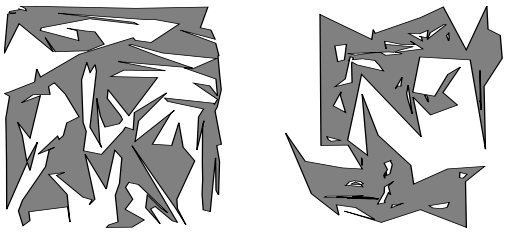
3 Reduced Convolution

We compute the reduced convolution set of segments filtering out features that cannot possibly contribute to the boundary of the Minkowski sum (see Section 1.1) and in particular complete holes (see Section 2). Then, we construct the arrangement induced by the reduced convolution set. Finally, we traverse the arrangement and extract the boundary of the Minkowski sum. We apply two different filters to identify valid holes in the Minkowski sum: (i) We ignore any face in the arrangement the outer boundary of which forms a cycle that is not properly oriented, as suggested in [2]. (ii) We ignore any face f, such that $(-P \oplus x)$ and Q collide, where $x \in f$ is a sampled point inside f, as suggested in [8]. We use axis-aligned bounding box trees to expedite the collision tests. After applying these two filters, only segments that constitute the Minkowski sum boundary remain.

In most cases, the reduced convolution method is faster than the full convolution method, as the induced arrangement has typically fewer cells. However, in degenerated cases with many holes in the Minkowski sum, the full convolution method is preferable over the reduced convolution method, as it avoids the costly collision-detection tests.

4 Experiments

We have conducted our experiments on families of randomly generated general and simple polygons from AGPLib [3]; examples are depicted in Figure 3a and 3b, respectively. All experiments were run on an *Intel Core 2 Duo P9600* CPU clocked at 2.53 GHz with 4 GB of RAM. For each instance size the figures show an average over 10 runs on different input.



(a) (b)

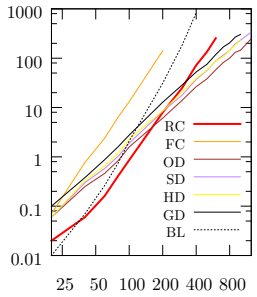
Figure 3: Randomly generated polygons: (a) simple, and (b) general.

First, we compared the running time of the implementations of all methods for simple polygons available in CGAL (for details, see [4, Section 9.1.2]), the new implementations, and Behar and Lien's implementation; see Figure 4a. Data points for instances not shown could not be computed due to memory limitations reached by the decomposition methods. The reduced convolution method consumed about ten times less time than the full convolution method for large instances, whereas the decomposition methods were the fastest for instances larger than 150 vertices.

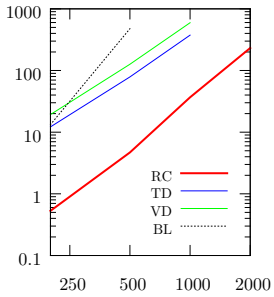
Secondly, we compared the running time of the implementations of the three new methods (i.e., RC, TD, and VD) and Behar and Lien's implementation on instances of general polygons with n vertices and $n/10$

holes; see Figure 4b. For each pair of polygons, one was scaled down by a factor of 1000, to avoid the effect of the hole filter in this experiment. Instances with more than 1000 vertices could not be processed by the decomposition methods due to memory limitations. For instances with more than 2000 vertices, the running time of the reduced convolution method exceeded 30 minutes. For all successful executions, the reduced convolution method consumed significantly less time than the two decomposition methods. Behar and Lien's implementation generally performs worse than our reduced convolution method.

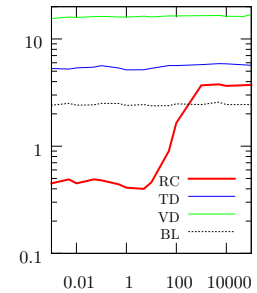
Thirdly, we compared the running time of the implementations above fed with a square of varying size (see the horizontal axis in Figure 4c and 4d) and with randomly generated polygons having 2000 vertices and 200 holes. While the running time of the reduced convolution method increased as the square grew due to an increase of the complexity of the intermediate arrangement, Behar and Lien's implementation exhibited constant running time, as it performs pairwise intersection testing. When applying the hole filter to our methods, the reduced convolution method consumed less time than all other methods. The two diagrams clearly show the impact of filtering holes.



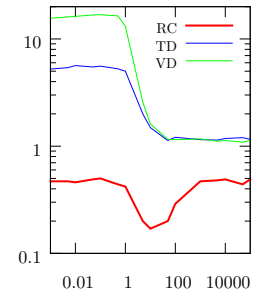
(a) MS of simple polygons.



(b) MS of general polygons.



(c) MS of general polygon and growing square (x-axis)—without hole filter



(d) MS of general polygon and growing square (x-axis)—with hole filter

Figure 4: Time in seconds for different methods to compute Minkowski sums for two polygons. Legend: (RC) reduced convolution; (FC) full convolution; (TD) constrained triangulation decomposition; (VD) vertical decomposition; (SD) small-side angle-bisector decomposition; (OD) optimal convex decomposition; (HD) Hertel-Mehlhorn decomposition; (GD) Greene decomposition; (BL) Behar and Lien's reduced convolution.

References

[1] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geom. Theory Appl.*, 21:39–61, 2002.

[2] E. Behar and J.-M. Lien. Fast and robust 2D Minkowski sum using reduced convolution. In *Proc. IEEE Conf. on Intelligent Robots and Systems*, 2011.

[3] M. C. Couto, P. J. de Rezende, and C. C. de Souza. Instances for the Art Gallery Problem, 2009. http://www.ic.unicamp.br/∼cid/Problem-instances/Art-Gallery.

[4] E. Fogel, D. Halperin, and R. Wein. *CGAL Arrangements and Their Applications, A Step by Step Guide.* Springer, Berlin Heidelberg, Germany, 2012.

[5] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Symp. Found. Comput. Sci.*, pp. 100–111, 1983.

[6] P. Hachenberger. Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica*, 55(2):329–345, 2009.

[7] E. E. Hartquist, J. Menon, K. Suresh, H. B. Voelcker, and J. Zagajac. A computing strategy for applications involving offsets, sweeps, and Minkowski operations. *Comput. Aided Design*, 31:175–183, 1999.

[8] A. Kaul, M. O'Connor, and V. Srinivasan. Computing Minkowski sums of regular polygons. In *Proc. 3rd Canadian Conf. on Comput. Geom.*, pp. 74–77, 1991.

[9] L. E. Kavraki. Computation of configuration-space obstacles using the fast fourier transform. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pp. 255–261, 1993.

[10] W. Li and S. McMains. A GPU-based voxelization approach to 3D Minkowski sum computation. In *Proc. 2010 ACM Symp. Solid Phys. Model.*, pp. 31–40. ACM Press, 2010.

[11] J.-M. Lien. A simple method for computing Minkowski sum boundary in 3D using collision detection. In H. Choset, M. Morales, and T. D. Murphey, editors, *Alg. Foundations of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pp. 401–415. Springer, 2009.

[12] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. on Comput.*, C-32:108–120, 1983.

[13] V. Milenkovic and E. Sacks. A monotonic convolution for Minkowski sums. *Int. J. of Comput. Geom. Appl.*, 17(4):383–396, 2007.

[14] The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 4.5 edition, 2014. `http://doc.cgal.org/latest/Manual/index.html`.

[15] G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. *Graphical Models*, 68(4):343–355, 2006.

[16] R. Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *Proc. 14th Annu. Eur. Symp. Alg.*, pp. 829–840, 2006.

On the Complexity of the Discrete Fréchet Distance under L_1 and L_∞

Omer Gold* Micha Sharir*

Abstract

We study the decision tree complexity of the discrete Fréchet distance (decision version) under the L_1 and L_∞ metrics over $\mathbb{R}^d$. While algorithms for the Euclidean (L_2) discrete Fréchet distance were studied extensively, the problem in other metrics such as L_1 and L_∞ seems to be much less investigated.

For the L_1 discrete Fréchet distance in $\mathbb{R}^d$ we present a $2d$-linear decision tree with depth $O(n \log n)$, for any constant d. For the L_∞ discrete Fréchet distance in $\mathbb{R}^d$ we present a 2-linear decision tree with depth $O(n \log n)$, for any constant d. We hope that these near-linear depth decision trees will motivate the study of the problem in these metrics and, in particular, will lead to the development of improved algorithms.

1 Introduction

The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the well-known Hausdorff distance as a metric for comparing parameterized shapes. This measure was introduced by Fréchet in 1906 [6].

Eiter and Mannila [5] introduced the *discrete Fréchet distance*, a variant also known as the *coupling distance*. They showed that this distance provides a good approximation for the Fréchet distance between curves, and provided a quadratic dynamic programming algorithm to compute it.

Since then many studies have been made about the *discrete* problem in the *Euclidean plane*: Agarwal et al. [1] showed a subquadratic algorithm with $O(n^2 \log\log n / \log n)$ runtime[1], Buchin et al. [3] showed an *algebraic computation tree* lower bound of $\Omega(n \log n)$, and Bringmann [2] recently showed that there is no algorithm with runtime $O(n^{2-\Omega(1)})$ (also known as "truly subquadratic time"), assuming the *Strong Exponential Time Hypothesis*. These bounds hold for computing the exact distance and for the decision version of the problem.

While much work has been made on the Euclidean discrete Fréchet distance, the problem in other metrics, such as L_1 and L_∞ has been much less investigated.

Buchin et al. [4] recently showed that the decision tree complexity of the *Euclidean* discrete Fréchet distance in the *plane* is[2] $\widetilde{O}(n^{4/3})$. This result is obtained by using a range searching technique of Katz and Sharir [9]. We will briefly review this result, and argue that, for the problem under the L_1 and L_∞ metrics in $\mathbb{R}^d$, the standard range searching approach does not seem capable of giving us the results we aim for, which we will establish using a different approach.

From now on, the term L_p discrete Fréchet distance refers to the *decision problem* of determining whether the discrete Fréchet distance with underlying norm L_p is at most some parameter $\varepsilon \geq 0$.

The contribution of this paper is given in the following theorems:

Theorem 1 *Given two polygonal curves P, Q in $\mathbb{R}^d$ with total complexity n (i.e., number of vertices), there is a $2d$-linear decision tree[3] with depth $O(n \log n)$ for the L_1 discrete Fréchet distance between P and Q, for any constant d.*

Theorem 2 *Given two polygonal curves P, Q in $\mathbb{R}^d$ with total complexity n, there is a 2-linear decision tree with depth $O(n \log n)$ for the L_∞ discrete Fréchet distance between P and Q, for any constant d.*

For Theorem 1 and Theorem 2, we generalize an observation originated in Fredman's 1976 work on the decision tree complexity of $(\min, +)$-matrix multiplication [7], a fundamental problem in **P**, known for being computationally equivalent to the APSP (all pairs shortest paths) problem in directed graphs with arbitrary real edge weights.

At the basis of Fredman's technique is the trivial (albeit ingenious) observation that $a + b < c + d$ iff $a - c < d - b$. This observation is often referred to as *Fredman's trick*. Fredman's trick was also liberally used by Grønlund and Pettie in their recent 3SUM breakthrough [8].

2 Fréchet Distance

The Fréchet distance is often illustrated by a man and a dog, each walking along a path (curve). The man

*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; {omergold, michas}@post.tau.ac.il

[1]For the decision version, they showed a bound of $O(n^2 \log\log n / \log^2 n)$. Both are in the word RAM model.

[2]The notation $\widetilde{O}(\cdot)$ hides poly-logarithmic factors.

[3]A k-linear decision tree is one in which each branching is based on a sign test of a linear expression with at most k terms.

has the dog on a leash. Each of them may choose their own speed and may stop but cannot walk backwards. Then the Fréchet distance is the length of the shortest leash that allows them to walk on their respective curves from beginning to end.

More formally, following [5] we define a curve as a continuous mapping $f : [0,1] \to V$, where (V, ρ) is a metric space. Given two curves $f : [0,1] \to V$ and $g : [0,1] \to V$, their Fréchet distance is defined as

$$\delta_F(f,g) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \rho(f(\alpha(t)), g(\beta(t))),$$

where α and β are arbitrary continuous nondecreasing functions from $[0,1]$ onto $[0,1]$.

When computing the Fréchet distance between arbitrary curves, one typically approximates the curves by polygonal curves. Eiter and Mannila [5] defined the *discrete Fréchet distance* between polygonal curves and showed it gives a good approximation to the Fréchet distance between them.

A polygonal curve with n edges is a curve $P : [0,1] \to V$, such that for each $i \in \{0,1,\ldots,n-1\}$, the restriction of P to the interval $\left[\frac{i}{n}, \frac{i+1}{n}\right]$ is affine. Since the Fréchet distance is invariant under reparametrization, we can assume a polygonal curve P to be given by the ordered list of its vertices, i.e., a sequence $P = (p_0, \ldots, p_n)$.

Let $P = (p_0, \ldots, p_n)$ and $Q = (q_0, \ldots, q_m)$ be two polygonal curves given by their ordered lists of vertices. A *coupling* $C = (c_0, \ldots, c_k)$ between P and Q is an ordered sequence of distinct pairs of vertices in P, Q, such that $c_0 = (p_0, q_0)$, $c_k = (p_n, q_m)$ and $c_r = (p_i, q_j) \Rightarrow c_{r+1} \in \{(p_{i+1}, q_j), (p_i, q_{j+1}), (p_{i+1}, q_{j+1})\}$. The *discrete Fréchet distance* between P and Q is

$$\delta_{dF}(P,Q) = \min_{C \text{ coupling}} \max_{(p_i,q_j)\in C} \rho(p_i, q_j).$$

Eiter and Mannila [5] showed that

$$\delta_F(P,Q) \le \delta_{dF}(P,Q) \le \delta_F(P,Q) + \max\{D(P), D(Q)\},$$

where $D(P)$ (resp., $D(Q)$) is the length of the longest edge in P (resp., Q). Thus, if we add vertices to the curves P, Q so that their edge lengths tend to zero, their discrete Fréchet distance will tend to their Fréchet distance.

Dynamic Programming Algorithm. Following [5], we quickly review the standard quadratic dynamic programming algorithm for the decision version of the discrete Fréchet distance, in a metric space (V, ρ).

Given two point sequences $A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_n)$, and a parameter $\varepsilon \ge 0$, the algorithm creates an $n \times n$ Boolean matrix M, whose rows and columns correspond to the points of A and B, respectively. The algorithm fills the matrix with values 0/1 row by row. Every cell $M_{i,j}$ in the matrix is filled by 1 iff both conditions hold:

1. At least one of the cells $M_{i-1,j}$, $M_{i,j-1}$, $M_{i-1,j-1}$ is filled with 1.
2. The distance $\rho(a_i, b_j)$ is at most ε.

Otherwise, $M_{i,j}$ is filled by 0. Intuitively, an entry $M_{i,j}$ is equal to 1 iff the pair (a_i, b_j) is reachable from the starting placement (a_1, b_1) of the trip with a "leash" of length ε. Otherwise, $M_{i,j}$ is equal to 0.

The runtime of the algorithm is quadratic and the number of input comparisons it does is also quadratic, as there are potentially n^2 distinct pairs of points (a_i, b_j) to check whether $\rho(a_i, b_j) \le \varepsilon$.

3 Decision Tree for the Euclidean Plane

Buchin et al. [4] showed a quadratic algebraic decision tree[4] with depth $O(n^{4/3} \log n)$ for the Euclidean discrete Fréchet distance in the plane.

The decision tree is based on invoking the quadratic dynamic programming algorithm following a preprocessing stage. All the input comparisons in the dynamic programming algorithm are made by checking if the distance of a point $a_i \in A$ from a point $b_j \in B$ is less than the fixed given parameter ε. The preprocessing stage will compute and store the answers for these pairwise distance queries in a Boolean matrix $T \stackrel{\text{def}}{=} (t_{ij})$, where $t_{ij} = 1$ if $\|a_i - b_j\|_2 \le \varepsilon$, otherwise $t_{ij} = 0$.

Given two point sequences A, B, with $|A| = n$, $|B| = m$, and a parameter $\varepsilon > 0$, denote, for each point $a \in A$, the circle of radius ε centered at a as c_a. A point $b \in B$ lies inside a circle c_a iff $\|a - b\|_2 \le \varepsilon$. We obtain a set C of n congruent circles (all of radius ε) and a set $P(= B)$ of m points.

Katz and Sharir [9] showed that one can compute a compact representation of the set of pairs of the form (c, p), where $p \in P$, $c \in C$, and p lies inside c, in $O((m^{2/3}n^{2/3} + m + n)\log n)$ time and space. This information suffices to construct T and invoke the dynamic programming algorithm without using further input comparisons.

Thus in total, when $|A| = |B| = n$, the number of input comparisons is $O(n^{4/3} \log n)$.

4 Decision Trees for L_1 and L_∞ in $\mathbb{R}^d$

Similar to the Euclidean case, range searching techniques can also be used for the problem under other metrics, for computing the pairwise distance queries in the decision tree. However, as we now show, these techniques, when routinely implemented, will give much weaker results than those stated in Theorem 1 and Theorem 2.

[4] Namely, each branching is a sign test of a quadratic expression.

The simpler case is for the L_∞ metric, for which the unit ball in $\mathbb{R}^d$ has the form of a d-dimensional hypercube. One can compute a d-dimensional range tree data structure for the points of A, in time $O(n \log^{d-1} n)$. For each point $b = (b_1, \ldots, b_d) \in B$, denote by c_b its corresponding d-sphere (under L_∞) of radius ε, centered at b. Clearly, $c_b = [x_1, y_1] \times [x_2, y_2] \times \cdots \times [x_d, y_d]$ is a d-dimensional hypercube.

For each $b \in B$, we query the range tree with its corresponding hypercube c_b. This will give us all the points of A that lie in c_b. Since for each interval of c_b, the query takes $O(\log n)$ time, the query for c_b takes $O(\log^d n)$ time. Using *fractional cascading*, this can be improved to $O(\log^{d-1} n)$ time. In total, this approach leads to a 2-linear decision tree of depth $O(n \log^{d-1} n)$.

For the L_1 metric, a similar approach will lead to a much weaker result. The unit ball under the L_1 metric forms a d-dimensional cross-polytope with 2^d facets. Thus, querying such a ball will require 2^d queries, each performing $O(\log n)$ $2d$-linear comparisons, resulting in a $2d$-linear decision tree of depth $O(n \log^{2^d} n)$.

The range searching data structure is appropriate also when the queries are not known in advance. Using Fredman's trick, we leverage the fact that in our case all the queries are known in advance, to obtain better decision trees.

The L_1 discrete Fréchet distance. We start by presenting a 4-linear decision tree with depth $O(n \log n)$ for the L_1 discrete Fréchet distance in $\mathbb{R}^2$, and then we explain how to modify it to obtain a $2d$-linear decision tree with depth $O(n \log n)$ for the problem in $\mathbb{R}^d$. This will prove Theorem 1.

The following property will allow us to apply *Fredman's trick* on pairwise distance queries under the L_1 norm.

For any real numbers $x, y, z \in \mathbb{R}$, $|x| + |y| \le z$ *if and only if* all the following inequalities hold.

$$x + y \le z, \qquad x - y \le z,$$
$$-x + y \le z, \qquad -x - y \le z.$$

Since the L_1 distance between a point $a_i = (x_i, y_i)$ and a point $b_j = (x_j, y_j)$ is defined by

$$\|a_i - b_j\|_1 = |x_i - x_j| + |y_i - y_j|,$$

the property above leads to the following observation.

Observation 1 *For $a_i = (x_i, y_i)$, $b_j = (x_j, y_j) \in \mathbb{R}^2$, $\|a_i - b_j\|_1 \le \varepsilon$ if and only if all the following inequalities hold.*

$$x_i + y_i \le x_j + y_j + \varepsilon,$$
$$x_i - y_i \le x_j - y_j + \varepsilon,$$
$$y_i - x_i \le y_j - x_j + \varepsilon,$$
$$-x_i - y_i \le -x_j - y_j + \varepsilon.$$

This observation is a sort of generalization of *Fredman's trick* for the L_1 distance between two points in the plane.

Recall that we are given two point sequences in the plane $A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_n)$, and a distance parameter ε. The following algorithm determines whether $\delta_{dF}(A, B) \le \varepsilon$.

1. Sort $D_1 \stackrel{\text{def}}{=} \{x_i + y_i,\ x'_j + y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
2. Sort $D_2 \stackrel{\text{def}}{=} \{x_i - y_i,\ x'_j - y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
3. Sort $D_3 \stackrel{\text{def}}{=} \{y_i - x_i,\ y'_j - x'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
4. Sort $D_4 \stackrel{\text{def}}{=} \{-x_i - y_i,\ -x'_j - y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
5. Using Observation 1, given the sorted orders on $D_1, \ldots, D_4$, construct the $n \times n$ Boolean matrix
$$T \stackrel{\text{def}}{=} (t_{ij}), \text{ where } t_{ij} = \begin{cases} 1 & \text{if } \|a_i - b_j\|_1 \le \varepsilon \\ 0 & \text{otherwise.} \end{cases}$$
6. Invoke the dynamic programming algorithm using T for the distance queries.

Steps 1–4 require $O(n \log n)$ comparisons. Using Observation 1, Step 5 requires no comparisons (on the raw data) at all, given the sorted orders on $D_1, \ldots, D_4$. Specifically, to test whether $\|a_i - b_j\|_1 \le \varepsilon$, we test the four corresponding inequalities from Observation 1. Each inequality test is resolved by the sorted orders on $D_1, \ldots, D_4$. Step 6 requires no comparisons, given the matrix T from Step 5. All comparisons are sign tests of 4-linear expressions. In total, the number of comparisons is $O(n \log n)$. The algorithm can be *implemented* to run in $O(n^2)$ time, using only $O(n \log n)$ input comparisons.

The algorithm can easily be extended to $\mathbb{R}^d$, by using additional sorting steps (similar to steps 1–4), and lead to a $2d$-linear decision tree with depth $O(n \log n)$. A generalization of Observation 1 to points $a_i = (x_{i_1}, \ldots, x_{i_d})$, $b_j = (x_{j_1}, \ldots, x_{j_d})$ in $\mathbb{R}^d$ leads to 2^d inequalities, each defined by a vector $\delta \in \{-1, 1\}^d$, and has the form

$$\sum_{k=1}^{d} \delta_k x_{i_k} \le \sum_{k=1}^{d} \delta_k x_{j_k} + \varepsilon.$$

Each such inequality is a $2d$-linear expression. Thus, for the same problem in $\mathbb{R}^d$, the algorithm has 2^d sorting steps, and all comparisons are sign tests of $2d$-linear expressions. This proves Theorem 1. □

The L_∞ discrete Fréchet distance. The previous algorithm can easily be modified (and simplified) for the L_∞ norm. As before, we first consider the problem in $\mathbb{R}^2$, and later extend it to $\mathbb{R}^d$. The L_∞ distance between a point $a_i = (x_i, y_i) \in \mathbb{R}^2$ and a point $b_j = (x_j, y_j) \in \mathbb{R}^2$ is defined by $\|a_i - b_j\|_\infty = \max\{|x_i - x_j|, |y_i - y_j|\}$. Hence,

$$\|a_i - b_j\|_\infty \leq \varepsilon \Leftrightarrow (|x_i - x_j| \leq \varepsilon) \wedge (|y_i - y_j| \leq \varepsilon).$$

Thus we obtain the following observation.

Observation 2 *For $a_i = (x_i, y_i)$, $b_j = (x_j, y_j) \in \mathbb{R}^2$, $\|a_i - b_j\|_\infty \leq \varepsilon$ if and only if all the following inequalities hold.*

$$x_i \leq x_j + \varepsilon, \qquad x_j \leq x_i + \varepsilon,$$
$$y_i \leq y_j + \varepsilon, \qquad y_j \leq y_i + \varepsilon.$$

This leads to the following variant of the previous algorithm, where the sets to be sorted are:

$$D_1 \stackrel{\text{def}}{=} \{x_i,\ x'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\},$$
$$D_2 \stackrel{\text{def}}{=} \{x'_j,\ x_i + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\},$$
$$D_3 \stackrel{\text{def}}{=} \{y_i,\ y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\},$$
$$D_4 \stackrel{\text{def}}{=} \{y'_j,\ y_i + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}.$$

Using Observation 2, given the sorted orders on $D_1, \ldots, D_4$, one can construct the Boolean matrix

$$T \stackrel{\text{def}}{=} (t_{ij}),\ \text{where } t_{ij} = \begin{cases} 1 & \text{if } \|a_i - b_j\|_\infty \leq \varepsilon \\ 0 & \text{otherwise,} \end{cases}$$

with no further comparisons. Now, one can invoke the dynamic programming algorithm and use T for the distance queries.

Similarly to the L_1 norm, the above algorithm uses $O(n \log n)$ input comparisons and can be implemented to run in $O(n^2)$ time. Each comparison is a sign test of a 2-linear expression.

Following a generalization of Observation 2 to points in $\mathbb{R}^d$, the algorithm can be extended to $\mathbb{R}^d$ by adding additional sorting steps. We have $2d$ sorting steps for the problem over $\mathbb{R}^d$, two for each coordinate. Each comparison will still be a 2-linear expression, independent of d. Thus in total we obtain a 2-linear decision tree with depth $O(n \log n)$ for the problem in $\mathbb{R}^d$, for any constant d. This proves Theorem 2. □

5 Discussion

An intriguing aspect of the presented results is the "large" gap we obtain between the nonuniform and the known uniform complexity of the problems.

For some archetypal problems in **P**, a gap of $\sqrt{n}$ was shown[5], starting with the $(\min, +)$-matrix multiplication [7], to the recent 3SUM and Zero Triangle results [8]. For the *Euclidean* discrete Fréchet distance in the plane, a gap of $n^{2/3}$ was noted above.

The quadratic time algorithm of Eiter and Mannila [5] can compute the discrete Fréchet distance in any metric space. For the L_1 and L_∞ versions, our $O(n \log n)$ decision trees give a gap of n. We hope that this "large" gap will motivate the study of the problem in these metrics. In particular, can one obtain a *truly subquadratic* algorithm for these problems? or on the other hand, does a similar result to the conditional lower bound of Bringmann [2] (for the *Euclidean* discrete Fréchet distance) can be obtained for the problem under metrics L_1 and L_∞?

Another open question is whether our decision trees are optimal. The $\Omega(n \log n)$ lower bound proof in [3] cannot be applied to the L_1 or L_∞ versions, as it exploits the *strict convexity* of the *Euclidean* plane.

References

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

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

[3] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? *Proc. 23rd Euro. Workshop Comput. Geom.*, pages 170–173, 2007.

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

[5] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report, TU Vienna, Austria, 1994.

[6] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Mathematico di Palermo*, 22:174, 1906.

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

[8] A. Grønlund and S. Pettie. Threesomes, degenerates, and love triangles. *Proc. 55th IEEE Annu. Sympos. Foundations of Computer Science*, pages 621–630, 2014.

[9] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26(5):1384–1408, 1997.

[5] We refer to such gaps by excluding $o(n^\varepsilon)$ factors, for any $\varepsilon > 0$.

A Middle Curve Based on Discrete Fréchet Distance*

Hee-Kap Ahn† Helmut Alt‡ Maike Buchin§ Ludmila Scharf¶ Carola Wenk‖

Abstract

Given a set of polygonal curves we seek to find a "middle curve" that represents the set of curves. We ask that the middle curve consists of points of the input curves and that it minimizes the discrete Fréchet distance to the input curves. We develop algorithms for three different variants of this problem.

1 Introduction

Consider a group of animals or people traveling together, several of which are GPS-tagged. Based on their trajectories, i.e., sequences of time-stamped positions, we want to compute a representation of a middle path taken by the group. Because sampled locations are more reliable than positions interpolated in between those, we seek a middle path consisting only of sampled locations. The middle path should be as close as possible to the path of the individuals, hence we ask for it to minimize the discrete Fréchet distance to these. The Fréchet distance [2] and the discrete Fréchet distance [4] are well-known distance measures, which have been used before in trajectory analysis.

We consider three variants of this problem, which we introduce now more formally for two curves. Given two point sequences P and Q, of length n and m respectively, and $\varepsilon > 0$, we wish to determine whether there exists a *middle curve* R consisting of points from $P \cup Q$ with $\max(d_F(R,P), d_F(R,Q)) \leq \varepsilon$, where d_F denotes the discrete Fréchet distance.

In the following definitions we assume that each point in R uniquely corresponds to a point in P or Q (in particular, if P and Q share points). We say the middle curve R is *ordered*, if any two points of P occurring in R have the same order as in P, likewise with points from Q. We say the middle curve R is *restricted*, if points on R are mapped to themselves in a matching realizing the discrete Fréchet distance. That is, consider a point p in R originating from P; In a matching between R and P realizing $d_F(R,P)$, p as a point of R is mapped to itself on P.

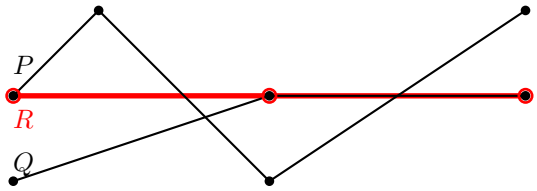


Figure 1: Example of a middle curve R of curves P, Q.

Related work. Several papers [3, 5] study the problem of finding a middle curve but without the restriction that the middle curve should consist of points of the input curves. Buchin et al. [3] restrict to use parts of edges of the input, and the aim is to always "stay in the middle" in the sense of a median. Har-Peled and Raichel [5] show that without any restrictions on the middle curve (i.e., neither using input vertices nor edges), a curve minimizing the Fréchet distance to k input curves can be computed in the k-dimensional free space using the radius of the smallest enclosing disk as "distance".

2-**Approximation.** A simple observation is that choosing any of the input curves is a 2-approximation to minimizing the distance (using the triangle inequality). Thus, we have a 2-approximation in constant time (not counting the time to output the points of the curve). Also, it is easy to give an example showing that this 2-approximation is tight.

Results. We develop algorithms for three variants of this problem (runtime for $k \geq 2$ curves of size at most n each):

1. An $O(n^k + n^2 \log n)$ time algorithm for computing an unordered middle curve,
2. An $O(n^{2k})$ time algorithm for computing an ordered middle curve,
3. An $O(n^k \log^{k-1} n)$ time algorithm for computing an ordered and restricted middle curve.

In the following, we will also call these three cases the "unordered, ordered, and restricted case". In the following sections, we present these algorithms. Due to space restrictions, we focus on describing the algorithms, omitting details and proofs.

*This work was partially supported by research grant AL 253/8-1 from Deutsche Forschungsgemeinschaft (German Science Association), and by the National Science Foundation under grant CCF-1301911.

†POSTECH, heekap@postech.ac.kr
‡Free University of Berlin, alt@mi.fu-berlin.de
§Ruhr University Bochum, maike.buchin@rub.de
¶Free University of Berlin, scharf@mi.fu-berlin.de
‖Tulane University, cwenk@tulane.edu

2 Algorithm for the unordered case

To solve the decision problem for the unordered case, we modify the algorithm for computing the discrete Fréchet distance of two curves [4] as follows. We search again for a path in the free space matrix. Now (in contrast to the original algorithm) we color a vertex (i, j) free iff there exists any vertex v from P or Q such that v has distance $\leq \varepsilon$ to both p_i and q_j. Then again we search for a monotone path in the free space matrix. For the computation problem, we label each vertex (i, j) with $\min_{v \in P \cup Q} \max(\|v - p_i\|, \|v - q_j\|)$, and search for a path minimizing the maximum label.

The runtime for searching the grid is (in both cases) $O(mn)$. To compute the vertex labels (0|1 or distances) takes $O(mn(m+n))$ time brute-force (i.e., for each vertex (i, j) test all $(m + n)$ possibilities for v in $O(1)$ time). For k curves of length at most n this takes $O(kn^{(k+1)})$ time in total. Next, we describe how to do this more efficiently. Here, we use a circular sweep to determine for each point p all points q such that (p, q) is free, i.e., there is some point v of P or Q which has distance $\leq \varepsilon$ to both p and q.

Constructing the free space matrix. For any $p \in P$:

1. Determine all disks of radius ε around points in $P \cup Q$ that contain p.
2. Determine the union U of those disks. This can be done by divide-and-conquer as follows: Since U is star-shaped its boundary ∂U is a sequence of circular arcs with vertices in between. We maintain the rays from p to these vertices sorted clockwise, say. Then it is easy to merge two boundaries of unions of $n/2$ disks into one of n disks.
3. Sort all points of Q around p in a clockwise fashion and merge them with the vertices of ∂U.
4. Perform a circular sweep around p with the points of Q and the vertices of ∂U as event points. During the sweep, compare each point $q \in Q$ encountered with the intersection point of the ray with the current circular arc of ∂U. Thus, it can be determined whether q is also in U. If so, mark the entry (p, q) in the free space matrix as "free".

Correctness. For the correctness of the algorithm, observe that for any pair (p, q) chosen in step 4 it must be true that q lies in one of the disks which contain p, and vice versa.

Runtime. One execution of step 1 takes time $O(m + n)$. In step 2, the complexity of ∂U is $O(n)$, see, e.g., [1]. The merging can be done in linear time, so the divide-and-conquer algorithm takes time $O((m + n) \log(m + n))$. Step 3 takes time $O(m \log m)$ for the sorting and $O(m + n)$ for the merging. Step 4 takes linear time. Since these steps are carried out for each point $p \in P$ the total runtime for setting up the free space matrix is $O(n(m + n) \log(m + n))$. Since the roles of P and Q can be exchanged we can achieve $O(\min(m, n)(m + n) \log(m + n))$ which is $O(mn \log(mn))$.

Output a middle curve. If in addition to a yes-answer for the decision problem also a covering sequence itself is wanted, each circular segment of ∂U should be labeled with the center point of its circle. This label is also entered into the free space matrix so that the sequence of labels of a monotone path gives a feasible unordered sequence for the middle curve.

Optimization problem. Solving the optimization problem can again be done by a binary search on the set of distances between pairs of points from $P \cup Q$ involving in each step the algorithm for the decision problem. This results in a $O(mn \log^2 mn)$ runtime.

Several curves. The decision algorithm can be extended to k curves $P^1, ..., P^k$. Then, having the outer loop for all points $p \in P^1$, say, in step 4 we determine which points $p_2 \in P^2, \ldots, p_k \in P^k$ lie inside U, as well. For all combinations $p, p_2, ..., p_k$ the corresponding entries in the k-dimensional free space matrix are marked as free. The runtime is $O(n_1 N \log N + M)$ where $N = \sum_{i=1}^{k} n_i$ and $M = \prod_{i=1}^{k} n_i$, which is only a minor improvement over the brute force algorithm with run time $O(N(N + M))$.

3 Dynamic programming for the ordered case

Now we present a dynamic programming algorithm for computing an ordered middle curve. As input we assume two sequences P, Q and we search for an ordered middle curve R. Let us denote by $P_i, 1 \leq i \leq n$, the "prefix" $(p_1, ..., p_i)$ of a sequence $P = (p_1, ..., p_n)$. P_0 is defined as the empty sequence.

Our dynamic programming algorithm operates with four-dimensional Boolean arrays of the form $X[i, j, k, l], 0 \leq i, k \leq n, 0 \leq j, l \leq m$, where $X[i, j, k, l]$ is `true` iff there exists an ordered sequence R from points in $P_i \cup Q_j$ with

$$\max(d_F(R, P_k), d_F(R, Q_l)) \leq \varepsilon.$$

We say in this case that R *covers* P_k and Q_l. Clearly, the decision problem has a positive answer iff $X[n, m, n, m]$ (or any $X[i, j, n, m]$) is true.

In order to determine the value of some $X[i, j, k, l]$ from entries of X with lower indices, we need more information, particularly, whether there is a covering sequence R in which the points p_i and q_j occur, and if they do, whether they occur in the interior or at the end of the sequence. To this end, the array X is the componentwise disjunction of seven Boolean arrays

$$X = A \vee B \vee C \vee D \vee E \vee F \vee G$$

with the meanings that a sequence R covering P_k and Q_l exists with the following properties, respectively:

$A[i,j,k,l]$: R contains neither p_i nor q_j.

$B[i,j,k,l]$: R contains p_i in its interior but does not contain q_j.

$C[i,j,k,l]$: R ends in p_i but does not contain q_j.

$D[i,j,k,l]$: R contains q_j in its interior but does not contain p_i.

$E[i,j,k,l]$: R ends in q_j but does not contain p_i.

$F[i,j,k,l]$: R contains q_j in its interior and ends in p_i.

$G[i,j,k,l]$: R contains p_i in its interior and ends in q_j.

Observe that R cannot contain both, p_i and q_j, in its interior (i.e. not at the end).

The entries of the arrays can be initialized or computed from entries with lower indices because of the following identities, which hold for each index $i,j,k,l \geq 1$, if that index minus 1 occurs in the formula and for all indices ≥ 0 otherwise.

$$\begin{array}{lcl} A[0,0,0,0] & = & \texttt{true} \\ A[0,0,k,l] & = & \texttt{false for } k \geq 1 \text{ or } l \geq 1 \\ A[i,0,k,l] & = & X[i-1,0,k,l] \\ A[0,j,k,l] & = & X[0,j-1,k,l] \\ A[i,j,k,l] & = & X[i-1,j-1,k,l] \end{array}$$

$$\begin{array}{lcl} B[i,0,k,l] & = & B[0,j,k,l] \;=\; \texttt{false} \\ B[i,j,k,l] & = & G[i,j-1,k,l] \vee B[i,j-1,k,l] \end{array}$$

The first equality is correct, since p_i must be at the end of R if no points from Q are available. In the second equality, $G[i,j-1,k,l]$ accounts for the case that R contains q_{j-1} (which then must be at the end) and $B[i,j-1,k,l]$ for the case that it does not.

In the following, let $cl(p,q)$ for points p and q denote the truth value for $\|p-q\| \leq \varepsilon$. These can be determined for all pairs of points in $P \cup Q$ by preprocessing. The following equalities for $C[i,j,k,l]$ are obtained by case distinction whether the final point p_i in the sequence R covers only p_k and q_l or also other points occurring previously in the sequences P_k and Q_l, respectively.

$$\begin{array}{lcl} C[i,j,0,l] & = & C[i,j,k,0] \;=\; C[0,j,k,l] \;=\; \texttt{false} \\ C[i,j,k,l] & = & cl(p_i,p_k) \;\wedge\; cl(p_i,q_l) \;\wedge \\ & & (A[i,j,k-1,l-1] \vee C[i,j,k-1,l-1] \\ & & \vee\, C[i,j,k-1,l] \vee C[i,j,k,l-1]) \end{array}$$

The entries of D and E can be determined analogously to the ones of B and C with the roles of p_i and q_j exchanged. The identities of F have similar explanations as the ones of C:

$$\begin{array}{lcl} F[0,j,k,l] & = & F[i,0,k,l] \;=\; F[i,j,0,l] \\ & = & F[i,j,k,0] \;=\; \texttt{false} \\ F[i,j,k,l] & = & cl(p_i,p_k) \wedge cl(p_i,q_l) \wedge \\ & & (D[i,j,k-1,l-1] \vee E[i,j,k-1,l-1] \\ & & \vee\, F[i,j,k-1,l] \vee F[i,j,k,l-1]) \end{array}$$

The entries of G can be determined analogously to the ones of F with the roles of p_i and q_j exchanged.

Runtime. The dynamic programm runs in time $O(n^2m^2)$ which is the size of each of the eight arrays.

Output a middle curve. Not only the existence of a covering sequence R, but R itself can be computed by setting a pointer for each array entry of the form $Y[i,j,k,l]$, which is set to true, to the 4-tupel(s) of indices at the right hand side of an equality that has made it true. Note that there can be an exponential number of valid middle curves.

Optimization problem. The value of $\max(d_F(R,P), d_F(R,Q))$ must be one of the distances between two points in $P \cup Q$. Therefore, the optimization problem can be solved by determining these distances, sorting them, and finding the correct value by binary search, invoking in each step the decision algorithm with the current value of ε. Altogether, this takes time $O(n^2m^2 \log(n+m))$.

Several Curves. The decision (and optimization) algorithm can be generalized to k sequences $P^1, ..., P^k$. The runtime in this case is $O(n_1^2 ... n_k^2)$ for constant k (but the number of arrays is $2^{k-1}(k+2)-1$).

4 Algorithm for the Restricted Case

Now the reparameterizations for minimizing $\max(d_F(R,P), d_F(R,Q))$ are restricted to map every vertex of R to itself in the input curve it originated from. This case allows for a more efficient dynamic program.

For this, we define arrays akin to Section 3. Let $X[i,j], 0 \leq i \leq n, 0 \leq j \leq m$, be `true` iff there exists an ordered sequence R from points in $P_i \cup Q_j$ with

$$\max(d_F(R,P_i), d_F(R,Q_j)) \leq \varepsilon,$$

with the restriction that any vertex of R is mapped to itself in the input curve it originated from. We say in this case that R *restrictively covers* P_i and Q_j. Clearly, the decision problem has a positive answer iff $X[n,m]$ is true.

Akin to Section 3 we can write X as a disjunction of three Boolean arrays

$$X = A \vee C \vee E$$

with the meanings that a sequence R covering P_i and Q_j exists with the following properties[1], respectively:

$A[i,j]$: R contains neither p_i nor q_j

$C[i,j]$: R ends in p_i (and may or may not contain q_j)

$E[i,j]$: R ends in q_j (and may or may not contain p_i)

[1] note that C here combines C and F in Section 3, and E combines E and G in Section 3

First we observe that

$A[i,j] \Leftrightarrow \exists\ i' < i$ and $j' < j$ such that $(C[i',j'] \wedge (i,j) \in U_P(i',j')) \vee (E[i',j'] \wedge (i,j) \in U_Q(i',j'))$

$C[i,j] \Leftrightarrow cl(p_i,q_j) \wedge$ there exist $i' < i$ and $j' < j$ such that $X[i',j'] \wedge (i',j') \in \hat{L}_P(i,j)$

$E[i,j] \Leftrightarrow cl(p_i,q_j) \wedge$ there exist $i' < i$ and $j' < j$ such that $X[i',j'] \wedge (i',j') \in \hat{L}_Q(i,j)$

Here, the *upper right* wedge $U_P(i',j')$ and the *lower left wedge* $L_P(i',j')$ represent subsets of point pairs (p_i,q_j) for which p_i and q_j are both close to $p_{i'}$. The upper right wedge consists of the *connected* set of such close point index pairs (i,j) for which $i' \le i$, $j' \le j$, and the set contains (i',j'). The lower left wedge consists of the *connected* set of such close point index pairs (i,j) for which $i \le i'$, $j \le j'$, and the point set contains (i',j').

Finally, we define the *extended lower left wedge* $\hat{L}_P(i',j')$ which, in addition to all points in the lower left wedge $L_P(i',j')$ also contains the points (i,j) immediately to the left or below, i.e., for which $(i+1,j)$, $(i,j+1)$, or $(i+1,j+1)$ is contained in $L_P(i',j')$. The definition of $U_Q(i',j'), L_Q(i',j'), \hat{L}_Q(i',j')$ is analogous, consisting of point pairs (p_i,q_j) for which p_i and q_j are both close to $q_{i'}$.

We compute X by incrementally adding true points using an enhanced bottom-up dynamic programming. In addition to storing the $(m+1) \times (n+1)$-array X, we also store the upper envelope $\bar{X}$ of all true points in X as a 1D array indexed by i. This will allow us to efficiently add reachable points to X. More specifically, we define $\bar{X}[i] = \max\{j \mid X[i,j] = true\}$. Note that X as well as $\bar{X}$ change during the dynamic programming, as more and more true points get added to X.

First, initialize all $X[i,j]$ to `false`, except for $X[0,0]$ which is set to `true`. Initialize $\bar{X}[0] = 0$, and $\bar{X}[i] = -1$ for all $i > 0$.

Then, for $i = 1$ to m, and for $j = 1$ to n, compute $X[i,j]$ (and update $\bar{X}$) as follows:

- If $X[i,j] \wedge cl(p_i,q_j)$: Add $U_P(i,j)$ and $U_Q(i,j)$ to X, together with a pointer to (i,j) that is labeled P or Q accordingly. An upper wedge is added to X by locating it in $\bar{X}$, updating the points in X that are above $\bar{X}$ by setting them to `true`, and finally updating $\bar{X}$. We refer to this as *updating X and $\bar{X}$ with the wedge*. This takes time proportional to the number of points updated.

- If $\neg X[i,j] \wedge cl(p_i,q_j)$: If $\hat{L}_P(i,j)$ intersects $\bar{X}$, update X and $\bar{X}$ with $L_P(i,j)$ and $U_P(i,j)$, and if $\hat{L}_Q(i,j)$ intersects $\bar{X}$, update X and $\bar{X}$ with $L_Q(i,j)$ and $U_Q(i,j)$. Also update pointers labeled with P or Q accordingly. The check can be done in constant time, and the update takes time proportional to the number of new points updated.

Correctness. For the correctness of the algorithm observe that if $X[i,j]$ holds because of $A[i,j]$, then it is marked when the last point of a covering is processed. If $X[i,j]$ holds by $C[i,j]$ or $E[i,j]$, then this is handled in the $\neg X[i,j] \wedge cl(p_i,q_j)$ case of the algorithm.

Runtime. By storing $\bar{X}$ in a binary search tree the algorithm runs in time $O(mn \log(\min(m,n)))$. For this, store $\bar{X}$ in a binary search tree sorted on i and augmented by the minimum value $\bar{X}[i]$ in a subtree rooted at a node. A rectangle with corners (i,j) and (u,r) can be queried by following the two paths to i and u. In between those paths process all subtrees with minimum smaller than r. Updating the values for $\bar{X}[i]$ and the minimum of these takes logarithmic time as well. Thus, it takes at most logarithmic time both to mark and to process an entry of X.

Several Curves. For $k > 2$ curves the algorithm works the same with a $k-1$ dimensional range tree for $\bar{X}$, and runtime $O(n^k \log^{k-1} n)$.

Acknowledgments. This work was initiated at the 17th Korean Workshop on Computational Geometry. We thank the organizers and all participants for the stimulating atmosphere. In particular we thank Fabian Stehn and Wolfgang Mulzer for discussing this paper.

References

[1] Pankaj K. Agarwal, János Pach, and Micha Sharir. State of the union (of geometric objects). In Jacob E. Goodman, Janos Pach, and Richard Pollack, editors, *Surveys on Discrete and Computational Geometry, Twenty Years Later*, volume 453 of *Contemporary Mathematics*, pages 9–48. ams, 2006.

[2] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.

[3] Kevin Buchin, Maike Buchin, Marc van Kreveld, Maarten Löffler, Rodrigo I. Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[4] Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical report, Technische Universität Wien, 1994.

[5] Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, January 2014.

Computing the Similarity Between Moving Curves*

Kevin Buchin† Tim Ophelders† Bettina Speckmann†

1 Introduction

A significant amount of algorithmic research in recent years has focused on the analysis of trajectories: sequences of time-stamped points which represent the movement of objects over time. Not all moving objects, however, can be reasonably represented as points. Here we hence go beyond this basic setting, by studying moving complex, non-point objects. Specifically, we focus on similarity measures for moving curves which can, for example, model changing coastlines, retreating glacier termini, or slithering snakes.

We base the similarity measures between moving curves on the Fréchet distance. We model a moving curve as a sequence of $T+1$ polylines, each of $P+1$ vertices. Consecutive polylines are interpolated to form a quadrilateral mesh of $P \times T$ quadrilaterals with parameters $(p,t) \in [0,P] \times [0,T]$.

The Fréchet distance is commonly used to determine the similarity between curves A and $B : [0,1] \to \mathbb{R}^n$. A natural generalization to more complex shapes uses the definition of Eq. 1 where A and B have type $X \to \mathbb{R}^n$.

$$D_{\text{fd}}(A,B) = \inf_{\mu: X \to X} \sup_{x \in X} \|A(x) - B(\mu(x))\| \qquad (1)$$

Here, $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ is a norm such as the Euclidean norm (L^2) or the Manhattan norm (L^1). The *matching* μ ranges over orientation-preserving homeomorphisms (possibly with additional constraints) between the parameter spaces of the shapes compared; as such, it defines a correspondence between the points of the compared shapes. Given one such matching we obtain a distance between A and B by taking the largest distance between any two corresponding points of A and B. The Fréchet distance is then the infimum of these distances taken over all possible matchings. For moving points or static curves, we have as parameter space $X = [0,1]$ and for moving curves or static surfaces, we have $X = [0,1]^2$. We define various similarity measures by imposing further restrictions on μ.

Related work. The Fréchet distance between point trajectories or polygonal curves can be computed in near-quadratic time [2]. The natural generalization to moving (parameterized) curves is to interpret the curves as surfaces parameterized over time and over the curve parameter. The Fréchet distance between surfaces is NP-hard [5], even for terrains [4]. In terms of positive algorithmic results for general surfaces the Fréchet distance is only known to be semi-computable [1].

When interpreting moving curves as surfaces it is important to take the different roles of the two surface parameters into account: the first is inherently linked to time while the other is linked to space. Depending on the application we do not want to treat these parameters equally. This naturally leads to restricted versions of the Fréchet distance of surfaces. For curves, restricted versions of the Fréchet distance have been previously considered [3, 6]. For surfaces we are not aware of similar results.

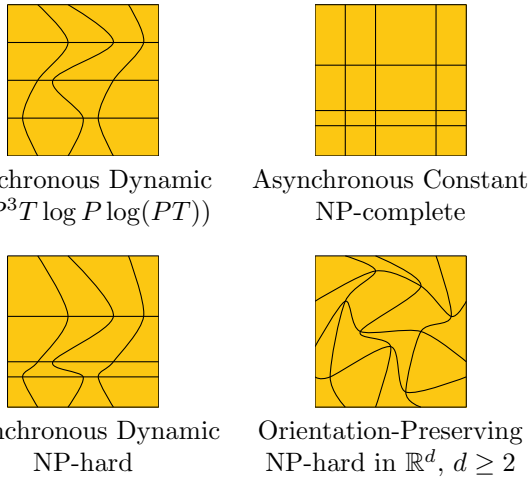


Figure 1: Time complexities; classes of matchings are illustrated as images of regular grids.

Results. We refine the Fréchet distance between surfaces to meaningfully compare moving curves. To do so, we restrict matchings to be one of several suitable classes. Here we often separate the matching into positional and temporal matchings. Representative matchings and running times for the classes considered are illustrated in Fig. 1.

2 Synchronous Dynamic Matchings

Synchronous dynamic matchings align timestamps under the identity matching, but the matching of positions may change continuously over time. Specifi-

*K. Buchin, T. Ophelders, and B. Speckmann are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.001.106 (K. Buchin) and no. 639.023.208 (T. Ophelders & B. Speckmann)

†Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands, [k.a.buchin|t.a.e.ophelders|b.speckmann]@tue.nl

cally, the matching is defined as $\mu(p,t) = (\pi_t(p), t)$. Here, $\mu(p,t) : [0,P] \times [0,T] \to [0,P] \times [0,T]$ is continuous, and for any t the matching $\pi_t : [0,P] \to [0,P]$ between the two curves is a nondecreasing surjection.

2.1 Freespace

Define the 3D freespace $\mathcal{F}_\varepsilon^{3D} \subseteq [0,P] \times [0,P] \times [0,T]$ by Eq. 2. Then the Fréchet distance is at most ε if and only if for some matching μ of the considered class, all points (x,y,t) with $\mu(x,t) = (y,t)$ lie in $\mathcal{F}_\varepsilon^{3D}$.

$$(x,y,t) \in \mathcal{F}_\varepsilon^{3D} \Leftrightarrow \|A(x,t) - B(y,t)\| \leq \varepsilon \quad (2)$$

Define cells $C_{x,y,t}$ of the freespace with $(x,y,t) \in \mathbb{N}^3$ by Eq. 3 as the freespace between two quadrilaterals.

$$C_{x,y,t} = [x,x+1] \times [y,y+1] \times [t,t+1] \cap \mathcal{F}_\varepsilon^{3D} \quad (3)$$

To determine the conditions under which some matching lies in the freespace, we derive some properties of freespace cells in Lemma 1.

Lemma 1 *Any cell $C_{x,y,t}$ has a convex intersection with any line parallel to the xy-plane or the t-axis.*

2.2 Freespace Partitions in 2D

Whereas previous algorithms for the decision problem of the Fréchet distance between curves compute a path through the freespace, we use a dual problem that extends to moving curves. We illustrate this dual approach in the fictional 2D freespace of Fig. 2. Here, any matching—such as the red path—must be an x- and y-monotone path from the bottom left to the top right corner and this matching must avoid all obstacles. Therefore each such matching divides the obstacles in two sets: those above, and those below the matching.

Suppose we are allowed to draw a directed edge from an obstacle a to an obstacle b if and only if any matching that goes over a must necessarily go over b. The key observation is that no matching exists if and only if such edges can form a path from the lower-right boundary to the upper-left boundary of the freespace. A few of these edges are drawn in black and gray. In the example, observe that if all obstacles were slightly larger, an edge could connect a blue and green obstacle, connecting the boundaries by the black edges.

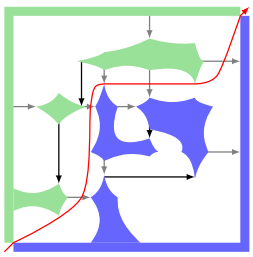


Figure 2: μ in 2D.

In contrast to the 2D freespace where the matching is a path, matchings form surfaces in the case of the 3D freespace. Such a surface again divides the obstacles in the freespace in two sets and can be punctured by a path connecting two boundaries. We shall formalize this approach for the 3D freespace ($\mathcal{F}_\varepsilon^{3D}$).

2.3 Freespace Partitions in 3D

Observe that any matching μ partitions obstacles into two sets, namely those above, and those below μ. Let O be the complete set of obstacles and $D \subseteq O$ be the obstacles below the matching. Then the upper boundary u of the freespace is never in D and the lower boundary d of the freespace is in D for any μ, see Fig. 3. The boundaries lie just outside the freespace. Let O' consist of all obstacles between the boundaries and let each obstacle be a connected subset of $\mathbb{R}^3$.

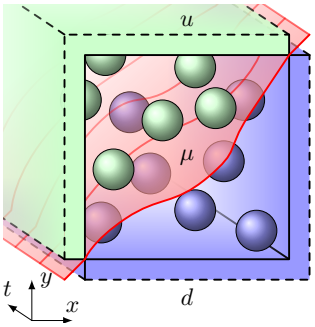


Figure 3: u, d and μ.

$$O = \{u,d\} \cup O' \text{ where } \bigcup O' = F \setminus \mathcal{F}_\varepsilon;$$

$$F = [0,P] \times [0,P] \times [0,T];$$

$$u = \{(x,y,t) \mid (x < 0 \wedge y > 0) \vee (x < P \wedge y > P)\};$$

$$d = \{(x,y,t) \mid (x > 0 \wedge y < 0) \vee (x > P \wedge y < P)\}.$$

Here, we use axes (x,y,t) and say that a point is below some other point if it has a smaller y-coordinate. Because each obstacle is a connected set and μ cannot intersect obstacles, a single obstacle cannot lie on both sides of the same matching. Because all matchings have $u \notin D$ and $d \in D$, a matching exists if and only if $\neg(d \in D \Rightarrow u \in D)$.

We compute a relation $\triangleright$ of elementary dependencies between obstacles, such that its transitive closure $\trianglerighteq$ has $d \trianglerighteq u$ if and only if $d \in D \Rightarrow u \in D$. Let $a \triangleright b$ if and only if $a \cup b$ is connected (a touches b) or there exists some point $(x_a, y_a, t_a) \in a$ and $(x_b, y_b, t_b) \in b$ with $x_a \leq x_b$, $y_a \geq y_b$ and $t_a = t_b$. By Lemmas 2 and 3, this choice of $\triangleright$ satisfies the required properties and by Theorem 4 we can use the transitive closure $\trianglerighteq$ of $\triangleright$ to solve the decision problem of the Fréchet distance.

Lemma 2 *If $a \trianglerighteq b$, then $a \in D \Rightarrow b \in D$.*

Lemma 3 *If $d \in D \Rightarrow u \in D$, then $d \trianglerighteq u$.*

Theorem 4 *The Fréchet distance is greater than ε if and only if $d \trianglerighteq u$ for ε.*

We choose the set of obstacles O' such that $\bigcup O' = F \setminus \mathcal{F}_\varepsilon$ and the relation $\triangleright$ is easily computable. Note that due to Lemma 1, each connected component contains a corner of a cell, therefore any cell in the freespace contains constantly many (up to eight) components of $F \setminus \mathcal{F}_\varepsilon$. As such, we can index the obstacles in O' by a grid point $(x,y,t) \in \mathbb{N}^3$ combined with one of the adjacent cells (with (x,y,t) as a corner).

$O' = \bigcup_{(x,y,t)\in\mathbb{N}^3\cap(F\setminus\mathcal{F}_\varepsilon)} O'_{x,y,t}$ where

$O'_{x,y,t} = \{o_{x,y,t,C} \mid \text{cell } C \text{ has } (x,y,t) \text{ as a corner}\}$,

$o_{x,y,t,C}$ is the maximal connected set with $(x,y,t) \in o_{x,y,t,C} \subseteq (F \setminus \mathcal{F}_\varepsilon) \cap C$.

Since obstacles in $O'_{x,y,t}$ touch at grid point (x,y,t), we treat them as a single obstacle $o_{x,y,t} = \bigcup O'_{x,y,t}$. Two obstacles $o_{x,y,t,C}$ and $o_{x',y',t',C}$ represent the same set of points if (x,y,t) is connected to (x',y',t') within C, but treat two such obstacles as distinct obstacles.

Each of the $O(P^2T)$ obstacles is now defined by a constant number of vertices. We therefore assume that for each pair of obstacles $(a,b) \in O^2$, we can decide in constant time whether $a \triangleright b$. For each obstacle a in a cell $C_{x,y,t}$, there can only be $O(P^2)$ obstacles b for which $a \triangleright b$; namely obstacles u, d, and those in cells $C_{x',y',t'}$ with $t' \in \{t-1, t, t+1\}$. Therefore we can compute the relation $\triangleright$ in $O(P^4T)$ time.

Testing whether $d \trianglerighteq u$ is equivalent to testing whether there exists a path from d to u in the directed graph $(O, \triangleright)$, which can be decided in $O(|\triangleright|)$ time using a depth first search. Thus, the decision problem for the Fréchet distance is solved in $O(|\triangleright|) = O(P^4T)$ time. There are many unnecessary edges in $\triangleright$ which we do not have to compute (see Theorem 5). To compute the exact Fréchet distance, the parametric search of Section 2.4 is applied to the decision problem.

Theorem 5 *The decision problem for the synchronous dynamic Fréchet distance is solvable in $O(P^3T \log P)$ time.*

2.4 Parametric Search

To give an idea of what the 3D freespace looks like, we have drawn the obstacles of the eight cells of the freespace between two quadrilateral meshes of size $P \times T = 2 \times 2$ in Fig. 4. Cells of the 3D freespace lie within cubes and have six faces and twelve edges. According to the axis to which they are parallel, we denote such edges by x-, y- or t-edges.

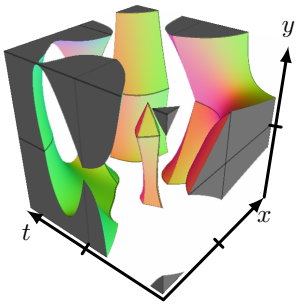


Figure 4: $[0,2]^3 \setminus \mathcal{F}_\varepsilon^{3D}$

We are looking for the minimum value of ε for which a matching exists. When increasing the value of ε, the relation $\triangleright$ becomes smaller since obstacles shrink. Critical values of ε occur when $\triangleright$ becomes smaller. Due to Lemma 1, all critical values involve an edge or an xt-face or yt-face of a cell, but never the internal volume, so the following critical values cover all cases.

a) The minimal ε such that $(0,0,t) \in \mathcal{F}_\varepsilon^{3D}$ and $(P,P,t) \in \mathcal{F}_\varepsilon^{3D}$ for all t.

b) An edge of $C_{x,y,t}$ becomes nonempty.

c) The endpoints of two y-edges (or two x-edges) of $C_{x,y,t}$ and $C_{x+i,y,t}$ (or $C_{x,y-j,t}$) align.

d) An endpoint of a t-edge of $C_{x,y,t}$ aligns with an endpoint of a t-edge of $C_{x+i,y-j,t}$.

e) An obstacle in $C_{x,y,t}$ stops overlapping with an obstacle in $C_{x+i,y,t}$ or $C_{x,y-j,t}$ when projected along the x- or y-axis.

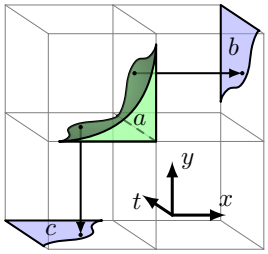


Figure 5: $a \triangleright b$ and $a \triangleright c$

We illustrate the need for critical values of type e) in Fig. 5. Here obstacle a overlaps with both obstacles b and c while the overlap in edges does not contribute to $\triangleright$. The critical values of types a), b) and c) resemble those in the paper by Alt and Godau [2]. The endpoints involved in the critical values of type a), b), c) and d) can be captured in $O(P^2T)$ functions.

We apply a parametric search [7] on them to find the minimum critical value $\varepsilon_{\text{abcd}}$ of type a), b), c) or d) for which a matching exists. This parametric search takes $O((P^2T + time_{\text{dec}}) \log(PT))$ time where $time_{\text{dec}} = O(P^3T \log P)$ is given by Theorem 5.

It is unclear how critical values of type e) can be incorporated in the parametric search directly. Instead, we enumerate and sort the $O(P^3T)$ critical values of type e) in $O(P^3T \log(PT))$ time. Using $O(\log(PT))$ calls to the decision algorithm, we apply a binary search to find the minimum critical value ε_{e} of type e) for which a matching exists. Finding ε_{e} then takes $O((P^3T + time_{\text{dec}}) \log(PT))$ time.

The synchronous dynamic Fréchet distance is then the minimum of $\varepsilon_{\text{abcd}}$ and ε_{e}. Because the decision problem takes $time_{\text{dec}} = O(P^3T \log P)$ time, the running time of Theorem 6 is achieved for the exact Fréchet distance.

Theorem 6 *The synchronous dynamic Fréchet distance can be computed in $O(P^3T \log P \log(PT))$ time.*

3 Hardness

We extend the synchronous dynamic class of matchings to the asynchronous dynamic class by allowing realignments of timestamps. Matchings of this class have the form $\mu(p,t) = (\pi_t(p), \tau(t))$ where π and τ are realign positions and timestamps and the positional matching π_t changes over time. In the more restricted asynchronous constant class $\mu(p,t) = (\pi(p), \tau(t))$ the

positional matching cannot change over time. The Fréchet distance is in NP for this class because piecewise linear π and τ exist whenever a matching exist.

Theorem 7 *Computing the Fréchet distance is in NP for the asynchronous constant class of matchings.*

Due to critical values of type e), it is unclear whether every asynchronous dynamic matching admits a piecewise-linear matching τ^* of polynomial size, which would mean that the asynchronous dynamic Fréchet distance is also in NP.

Computing the Fréchet distance is NP-hard for both classes by a reduction from 3-SAT. The idea behind the construction is illustrated in the two height maps of Fig. 6. The height maps represent quadrilateral meshes embedded in $\mathbb{R}^1$ and correspond to a single clause in a 3-CNF formula of four variables.

We distinguish valleys (dark), peaks (white on A, yellow on B) and ridges (denoted X_i, F_i and T_i). Observe that to obtain a low Fréchet distance of $\varepsilon < 3$, the n-th valley of A must be matched with the n-th valley of B. Moreover, each ridge X_i must be matched with F_i or T_i and each peak of A must be matched to a peak of B. Note that even for asynchronous dynamic matchings, if X_i is matched to F_i it cannot be matched to T_i and vice-versa because the (red) valley separating F_i and T_i has distance 3 from X_i.

Consider a 3-CNF formula with n variables and m clauses, then A and B consist of m clauses along the t-axis and n variables ($X_1 \dots X_n$ and $F_1, T_1 \dots F_n, T_n$) along the p-axis. The k-th clause of A is matched to the k-th clause of B due to the elevation pattern on the far left ($p = 0$). This means that the peaks of A are matched with peaks of the same clause on B and for these peaks have the same timestamp because $\tau(t)$ does not depend on p. For each clause, there are three rows (timestamps) of B with peaks on the ridges. On each such timestamp, exactly one ridge (depending on the disjuncts of the clause) does not have a peak. Specifically, if a clause has X_i or $\neg X_i$ as its k-th disjunct, then the k-th row of that clause has no peak on ridge F_i or T_i, respectively. By Theorem 10, it is then NP-hard to approximate the Fréchet distance within a factor 1.5.

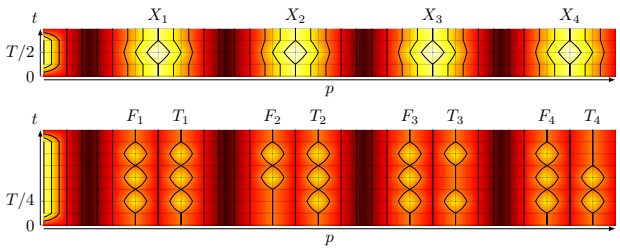


Figure 6: Meshes A (top) and B (bottom) in $\mathbb{R}^1$ (indicated by color). Their Fréchet distance is two isolines if $(X_2 \vee \neg X_3 \vee \neg X_4)$ is satisfiable and three otherwise.

Under the class of orientation-preserving homeomorphisms (restricted by aligning the four corners between of the meshes), we can embed the meshes in $\mathbb{R}^2$ and ensure that all points on A of the same timestamp are matched to similar timestamps of B and Theorem 11 follows.

Lemma 8 *The Fréchet distance between two such moving curves is at least* 3 *if the corresponding 3-CNF formula is unsatisfiable.*

Lemma 9 *The Fréchet distance between two such moving curves is at most* 2 *if the corresponding 3-CNF formula is satisfiable.*

Theorem 10 *No polynomial time algorithm can approximate the asynchronous constant or asynchronous dynamic Fréchet distance between two quadrilateral meshes in* $\mathbb{R}^1$ *within a factor* 1.5 *unless P=NP.*

Theorem 11 *No polynomial time algorithm can approximate the orientation-preserving Fréchet distance between quadrilateral meshes in* $\mathbb{R}^2$ *under the maximum norm within a factor* 1.5 *unless P=NP.*

References

[1] H. Alt and M. Buchin. Can we compute the similarity between surfaces? *Discrete Comput. Geom.*, 43(1):78–99, 2010.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(01n02):75–91, 1995.

[3] K. Buchin, M. Buchin, and J. Gudmundsson. Constrained free space diagrams: a tool for trajectory analysis. *International Journal of Geographical Information Science*, 24:1101–1125, 2010.

[4] K. Buchin, M. Buchin, and A. Schulz. Fréchet distance of surfaces: Some simple hard cases. In *Algorithms–ESA 2010*, pages 63–74. Springer, 2010.

[5] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions.* PhD thesis, Berlin, Freie Universität Berlin, Diss., 1998, 1999.

[6] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Fréchet distance with speed limits. *Comput. Geom. Theory Appl.*, 44(2):110–120, 2011.

[7] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

Exact solutions for the continuous 1.5D Terrain Guarding Problem

Stephan Friedrichs* Michael Hemmer† Christiane Schmidt‡

Abstract

In the *NP*-hard continuous 1.5-dimensional Terrain Guarding Problem (TGP) we are given an x-monotone chain (the *terrain* T) and ask for the minimum number of point guards (located anywhere on T), such that all points of T are covered. We recently gave guard candidate and witness sets of polynomial size, G and W, such that there exists a minimum-cardinality guard cover $G^* \subseteq G$ that covers T, and when these guards monitor all points in W, all of T is guarded. This leads to a PTAS as well as an (exact) IP formulation for TGP. In this paper, we significantly reduce the size of G and W, allowing us to propose an algorithm for reliably finding exact, optimal solutions for instances with 100000 vertices within seconds.

1 Introduction

Let a *terrain* T denote an x-monotone chain defined by its vertices $V = \{v_1, \dots, v_n\}$. It has edges $E = \{e_1, \dots, e_{n-1}\}$ with $e_i = \overline{v_i v_{i+1}}$. For $p, q \in T$, we write $p < q$ if p is left of q.

A point $p \in T$ *sees* or *covers* $q \in T$ iff $\overline{pq}$ is nowhere below T. $\mathcal{V}(p)$ is the *visibility region* of p with $\mathcal{V}(p) = \{q \in T \mid p \text{ sees } q\}$. $\mathcal{V}(p)$ is not necessarily connected, and can be considered as the union of $\mathrm{O}(n)$ maximal subterrains, compare Figure 1. We say that $q \in \mathcal{V}(p)$ is *extremal* in $\mathcal{V}(p)$, if q has a maximal or minimal x-coordinate within its connected component of $\mathcal{V}(p)$. For $G \subseteq T$ we abbreviate $\mathcal{V}(G) := \bigcup_{g \in G} \mathcal{V}(g)$. A set $G \subseteq T$ with $\mathcal{V}(G) = T$ is named a *(guard) cover* of T. In this context, $g \in G$ is referred to as *guard*.

Definition 1 (Terrain Guarding Problem) *For a terrain T and sets of guard candidates and witnesses, G and W, the* Terrain Guarding Problem (TGP), $\mathrm{TGP}(G, W)$, *asks for a minimum-cardinality $G^* \subseteq G$ such that $W \subseteq \mathcal{V}(G^*)$. We assume $W \subseteq \mathcal{V}(G)$, i. e., that $\mathrm{TGP}(G, W)$ has a feasible solution.*

$\mathrm{TGP}(T, T)$ is the continuous TGP and $\mathrm{TGP}(V, T)$ that with vertex guards. Motivation for terrain guarding is the placement of street lights or security cameras along roads [8], and the optimal placement of antennas for line-of-sight communication networks [1].

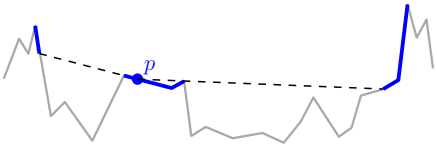


Figure 1: The visibility region $\mathcal{V}(p)$ of point p (blue).

1.1 Related Work

The terrain guarding problem is closely related to the well known Art Gallery Problem where the objective is to find a minimum cardinality guard set that covers a given polygon. See, e. g., O'Rourke [12] for a detailed survey on classical results and de Rezende et al. [5] for recent computational developments.

For the terrain guarding problem the focus was on approximation algorithms [1, 3, 10, 6], because *NP*-hardness was generally assumed but only shown in 2010 by King and Krohn [11]. In 2009, Gibson et al. [8] showed that the discrete terrain guarding problem $\mathrm{TGP}(G, W)$, where G and W are finite subsets of the terrain T, allows a *polynomial time approximation scheme* (PTAS) based on local search. For the continuous problem we recently [7] gave polynomial size guard and witness sets, G and W, such that there exists an optimal guard cover $G^* \subseteq G$ that covers T. This leads to a PTAS as well as an (exact) IP formulation for the continuous terrain guarding problem.

Our Contribution After summarizing the essentials of our discretization [7] (Section 2) we derive a filter that significantly reduces the size of constructed sets G and W (Section 3). This drastically reduces the overall complexity, making it possible to devise a reliable algorithm producing optimal solutions for instances of 100000 vertices within seconds on a desktop PC; see Section 4.

2 Discretization

This section summarizes our discretization from [7]. Section 2.1 shows how to construct a finite witness set $W(G)$ from a given finite guard candidate set $G \subset T$ such that any feasible solution of $\mathrm{TGP}(G, W(G))$ is feasible for $\mathrm{TGP}(G, T)$ as well. Section 2.2 discusses a

*Max Planck Institute for Informatics, Saarbrücken, Germany. E-mail: sfriedri@mpi-inf.mpg.de.

†TU Braunschweig, IBR, Algorithms Group, Braunschweig, Germany. E-mail: mhsaar@gmail.com

‡The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem. E-mail: cschmidt@cs.huji.ac.il. Supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

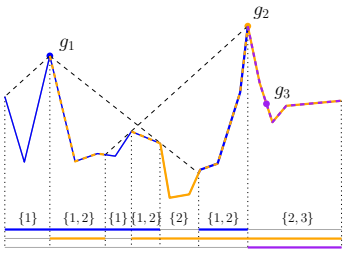


Figure 2: Visibility overlay of three guards.

finite set of guards U that allows minimum-cardinality coverage of T. An IP formulation for an exact solutions as well as a PTAS built upon [8] follows.

2.1 Witnesses

Suppose we are given a terrain T and a finite set $G \subset T$ of guard candidates with $\mathcal{V}(G) = T$. We provide a witness set $W(G)$ such that $\mathrm{TGP}(G,T) = \mathrm{TGP}(G,W(G))$ by computing the overlay of all visibility intervals of all guards in G as indicated in Figure 2. It forms a subdivision consisting of *features* f (either maximal intervals or end points). Every point in f is seen by the same set of guards

$$G(f) = \{g \in G \mid f \subseteq \mathcal{V}(g)\}\,. \tag{1}$$

It is thus sufficient to cover one representative witness w_f in each feature f of the overlay:

$$W_T(G) = \{w_f \mid f\} \tag{2}$$

which we refer to as *trivial witnesses*. Similar to shadow atomic visibility polygons [5], we can reduce the number of witnesses by only using *shadow witnesses*, i. e., witnesses in features f, such that $G(f) \subseteq G(f')$ for the neighboring features f' of f:

$$W_S(G) = \{w_f \mid f,\, G(f) \text{ is inclusion-minimal}\}\,. \tag{3}$$

The asymptotic complexity remains unchanged: $|W_S(G)|, |W_T(G)| \in \mathrm{O}(n|G|)$ [7], but in practice, $|W_S(G)| \ll |W_T(G)|$. So picking $W(G) := W_S(G)$ does make a difference; see Section 4.

2.2 Guard Positions

Throughout this section, let T be a terrain, V its vertices and E its edges; let $C \subset T$ be some finite, possibly optimal, guard cover of T. Moreover, let U be all vertices and their visibility regions' extremal points:

$$U := V \cup \bigcup_{v \in V} \{p \mid p \text{ is extremal in } \mathcal{V}(v)\}\,. \tag{4}$$

It is easy to see that U has cardinality $\mathrm{O}(n^2)$ [1]. We show that for any cover C it is always possible to move guards in $C \setminus U$ to a neighboring point in U without losing coverage. In particular, this is possible for an optimal guard cover.

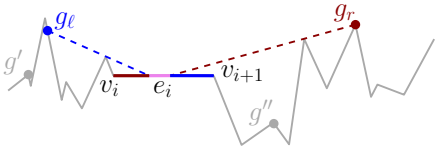


Figure 3: The edge e_i is critical w. r. t. g_ℓ and g_r. The right (left) part of e_i is seen by g_ℓ (g_r) only.

First observe that we can not lose coverage for an edge e that is entirely covered by a guard $g \in C \setminus U$ if we move g to one of its neighbors in U.

Lemma 2 ([7]) *Let $g \in C \setminus U$ be a guard that covers an entire edge $e_i \in E$. Then u_ℓ, u_r, the U-neighbors of g with*

$$\begin{aligned} u_\ell &= \max\{u \in U \mid u < g\} \\ u_r &= \min\{u \in U \mid g < u\} \end{aligned} \tag{5}$$

each entirely cover e_i, too.

It remains to consider edges that are not entirely covered by a single guard; see Figure 3:

Definition 3 (Critical Edge) *$e \in E$ is a* critical edge w. r. t. *g in the cover C if $C \setminus \{g\}$ covers some part of, but not all of, e.*

Definition 4 (Left-Guard/Right-Guard) *$g \in C$ is a* left-guard (right-guard) *of $e_i \in E$ if $g < v_i$ ($v_{i+1} < g$) and e_i is critical w. r. t. g. We call g* left-guard (right-guard) *if it is a left-guard (right-guard) of some $e \in E$.*

The following Lemma, which is a stronger variant than one given in [7], shows that we can move a left-guard $g \in C \setminus V$ to its left neighbors in V.

Lemma 5 ([7]) *Let C be some finite cover of T, $g \in C \setminus V$ be a left- but no right-guard, and let v_ℓ be the left neighbor in V of g. Then*

$$C' = (C \setminus \{g\}) \cup \{v_\ell\} \tag{6}$$

is a guard cover of T.

Proof. Since g is a left-guard there must exists a corresponding right-guard g_r as depicted in Figure 4. g is dominated to its left by g_r. Moreover, g is dominated to its right by v_ℓ; see Figure 5. Hence, we can replace g by v_ℓ. □

The following lemma is also a stronger version of one given in [7]. It states that no guard that is not on a vertex can be a left- and right-guard at the same time. We skip the proof, but the idea is given in Figure 4.

Lemma 6 *Let C be some finite cover of T. No $g \notin V$ is both a left- and a right-guard.*

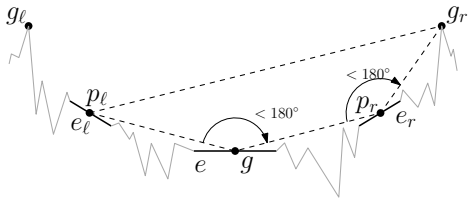


Figure 4: Left-guard g is dominated to its left by right-guard g_r. If g would also be a right-guard, then left-guard g_ℓ would dominate g to its right as well, implying that g is not necessary at all.

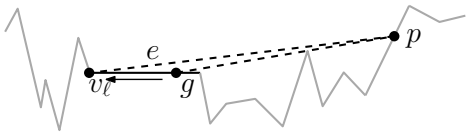


Figure 5: v_ℓ dominates g to its right.

Hence, there is no guard that is left- and right-guard at the same time. Moreover, by Lemma 5 we know that left/right-guards can even be moved to their left/right neighbor in V. Only *free guards*, i.e., guards that are neither left- nor right-guards, require the set U; see Lemma 2. It follows:

Theorem 7 ([7]) *Let T be a terrain, $C \subset T$ a finite guard cover of T, possibly of minimum cardinality, and consider U as defined in Equation (4). Then there exists a guard cover $C' \subseteq U$ of T with $|C'| = |C|$.*

2.3 Complete Discretization and IP

Combining the above results implies:

Theorem 8 ([7]) *Let T be a terrain, U and $W_S(U)$ as defined in Equations (4) and (3). Then: If C is solution of $\mathrm{TGP}(U, W_S(U))$ of minimal cardinality, C is also an optimal solution of $\mathrm{TGP}(T, T)$.*

As described in [7], combining this with [8] directly implies the existence of a PTAS for $\mathrm{TGP}(T, T)$. Also, an IP formulation immediately follows which is the basis for our algorithm:

$$\min \sum_{g \in U} x_g \tag{7}$$

$$\text{s.t.} \sum_{g \in \mathcal{V}(w) \cap U} x_g \geq 1 \quad \forall w \in W_S(U) \tag{8}$$

$$x_g \in \{0, 1\} \qquad \forall g \in U \tag{9}$$

Observation 1 ([7]) *: The set of guard candidates U has cardinality $\mathrm{O}(n^2)$, the witness set $W_S(U)$ has cardinality $\mathrm{O}(n^3)$.*

3 Reducing the Number of Guard Candidates

For practical purposes, we need to reduce the size of U. The reason for that is twofold: (1) Fewer guards need less memory and reduce the number of variables in the IP. (2) Having fewer guards automatically generates less witnesses, further reducing IP size. We present two filtering mechanisms.

3.1 Free-guard Filter

Consider all guard candidates in the interior of an edge e, $U_e = e \cap (U \setminus V)$ and recall that when moving a guard across $u \in U_e$, a vertex becomes visible or invisible. The set of edges *entirely* seen by u, E_u, defines a partial order on U_e: u is inclusion-maximal, if $E_{u'} \subseteq E_u$ for all $u' \in U_e$. We show it suffices to consider the inclusion-maximal guard candidates w. r. t. this ordering.

Theorem 9 *Let U'_e be the set that only contains inclusion-maximal guard positions of U_e w. r. t. entire edges. U'_e contains all guard candidates in the interior of e that are required for an optimal cover of T.*

Proof. By Lemma 5 a left/right-guard can even be moved to its left/right neighbor in V. By Lemma 6, it remains to consider free guards, that is, guards that are only responsible for covering entire edges, which can obviously be replaced by an inclusion-maximal alternative. □

Hence, we can filter out candidates in U that are not inclusion-maximal. By remembering which vertex generated which point in U, this filter can be applied within one pass over the set U, specifically, without calculating visibility regions for any point in $U \setminus V$.

3.2 Domination Test

Obviously, for different $g, g' \in U$ with $\mathcal{V}(g') \subseteq \mathcal{V}(g)$, $U \setminus \{g'\}$ still admits an optimal guard cover. It takes $\mathrm{O}(n|U|^2) = \mathrm{O}(n^5)$ time to filter out all dominated guards from U. Our implementation applies it only to neighboring g, g', which takes $\mathrm{O}(n|U|)$ time and still eliminates many guard candidates.

4 Implementation and Experiments

We implemented the following algorithm to determine an optimal solution of $\mathrm{TGP}(T, T)$: (i) Compute U as in Equation (4), optionally (i-a) filter U as in Section 3.1, and, (i-b) filter U as in Section 3.2. (ii) Determine shadow or trivial witnesses as in Equation (3) or (2) from the remaining candidates in U. (iii) Solve (7) – (9).

The geometric part, phase (i)-(ii), of our implementation is based on CGAL [2] and follows the exact geometric computation (EGC) paradigm. Specifically, it uses the new algorithms for visibility in terrains presented in [9], which are part of the upcoming Visibility

n	DEFAULT	NODOM	TRIVIAL	NOFREE
10^3	100%	100%	100%	100%
10^4	100%	100%	75%	100%
10^5	100%	99%	41%	48%

Table 1: Optimal solution rates.

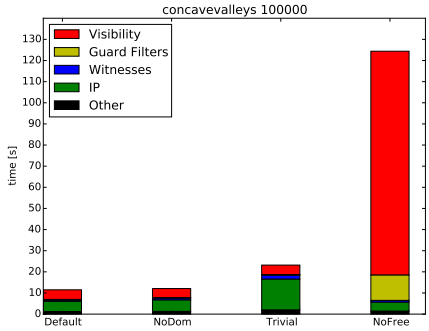


Figure 6: CPU-time spent by subroutine.

package of CGAL. Phase (iii) uses CPLEX-12.1.0 [4] for solving the IPs.

The tests were run on Intel Core i7-3770 CPUs with 3.4 GHz, with imposed time- and memory limits of 900 s and 12 GB. For each $n \in \{1000, 10000, 100000\}$, we tested 80 instances with n vertices. In order to determine the effect of each optional step, we tested the following configurations of the algorithm: DEFAULT runs both filters and shadow witnesses; TRIVIAL applies all filters, but uses trivial witnesses; NOFREE applies only filter (i-b), using shadow witnesses; NODOM applies only filter (i-a), and uses shadow witnesses. Hence, we test the version with all features enabled and disable one feature at a time.

Table 1 indicates how effective our algorithm configurations were within the allotted limits: DEFAULT and NODOM clearly outperform TRIVIAL and NOFREE. Thus, even though our filtering mechanisms do not reduce the asymptotic complexity of U and $W(U)$, they save our algorithm both memory and time. On average, DEFAULT eliminated 98.4% of U.

Figure 6 shows how much time was spent in which phase. It suggests that the time spent removing adjacent dominated guards roughly balances with the benefits from a reduced IP size. The additional amount of witnesses in TRIVIAL increases IP solution times. Not filtering the free guard candidates in NOFREE deals a devastating blow to performance: It imposes many more visibility calculations and, paradoxically, increases filtering times, caused by the sheer amount of guards that have to be checked for domination.

Note that, due to efficient implementation, the bottleneck of our algorithm is not the *NP*-hard IP solver phase, nor a time problem at all; 92% of the unsuccessful runs (of all configurations and input sizes) terminated due to running out of memory, not time. This is owed to the fact that, even with good filters, many visibility regions have to be stored, all of them with exact (not floating-point) coordinates—another indicator why it is important to filter guards. Especially the free guard filter from Theorem 9 has a very positive effect, because it can be run before calculating the non-vertex-guards' visibility regions.

5 Conclusion

We significantly reduced the the number of guard candidates in our discretization [7] of TGP(T, T). This allows us to reliably find exact, optimal guard covers for 100000-vertex terrains within seconds on a desktop computer.

References

[1] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5D terrain guarding. *SIAM J. Comput.*, 36(6):1631–1647, 2007.

[2] CGAL (Computational Geometry Algorithms Library). `http://www.cgal.org/`.

[3] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[4] IBM ILOG CPLEX Optimization Studio. `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`.

[5] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering Art Galleries. *ArXiv e-prints*, Oct. 2014.

[6] K. M. Elbassioni, E. Krohn, D. Matijevic, J. Mestre, and D. Severdija. Improved approximations for guarding 1.5-dimensional terrains. *Algorithmica*, 60(2):451–463, 2011.

[7] S. Friedrichs, M. Hemmer, and C. Schmidt. A PTAS for the continuous 1.5d terrain guarding problem. In *In: 26th Canadian Conference on Computational Geometry*, pages 367–373, 2014.

[8] M. Gibson, G. Kanade, E. Krohn, and K. Varadarajan. An approximation scheme for terrain guarding. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 5687 of *LNCS*, pages 140–148. 2009.

[9] A. Haas and M. Hemmer. Efficient Algorithms and Implementations for Visibility in 1.5D Terrains. In *EuroCG*, 2015.

[10] J. King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In J. R. Correa, A. Hevia, and M. A. Kiwi, editors, *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 629–640. Springer, 2006.

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

[12] J. O'Rourke. *Art Gallery Theorems and Algorithms*. International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

Efficient Algorithms and Implementations for Visibility in 1.5D Terrains

Andreas Haas* Michael Hemmer*

Abstract

Visibility is a well studied problem in computational geometry as it is used as a basic building block by many algorithms. In this paper we focus on visibility in 1.5D terrains. We report on new implementations and corresponding experimental evaluations for an extended version of the sweep line algorithm recently presented by Löffler et al. as well as a variant incorporating ideas of the Triangular Expansion algorithm for polygons of Bungiu et al. Our algorithms are currently submitted as an extension of the upcoming visibility package of the Computational Geometry Algorithms Library (CGAL).

1 Introduction

Let T denote a *terrain*, that is, an x-monotone chain defined by its vertices $V = \{v_1, \ldots, v_n\}$. It has edges $E = \{e_1, \ldots, e_{n-1}\}$ with $e_i = \overline{v_i v_{i+1}}$. A point p placed anywhere on or above T *sees* $q \in T$ iff $\overline{pq}$ is nowhere below T. $\mathcal{V}(p)$ is the *visibility map* of p with $\mathcal{V}(p) = \{q \in T \mid p \text{ sees } q\}$. $\mathcal{V}(p)$ is not necessarily connected, and can be considered as the union of $\mathrm{O}(n)$ maximal subterrains. For a set of m view points P we abbreviate $\mathcal{V}(P) := \bigcup_{p \in P} \mathcal{V}(p)$, see Figure 1 for an example of a visibility map of two points.

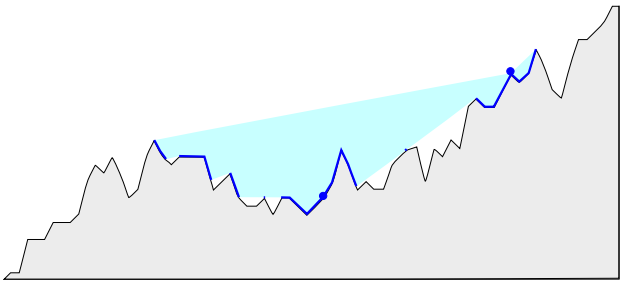


Figure 1: The visibility map of two viewpoints (blue dots).

We are interested in an efficient computation of $\mathcal{V}(P)$ for $|P| = m \geq 1$, which is an important building block in our ongoing efforts [7] to provide optimal guard covers of 1.5D terrains. A problem that occurs in the context of the placement of street lights or security cameras along roads [9], or the optimal placement of antennas for line-of-sight communication networks [2].

*Department of Computer Science, TU Braunschweig, Germany. andreas.haas86@gmail.com, mhsaar@gmail.com

Setting up the integer program (IP) for the terrain guarding problem (TGP) described in [6] by Friedrichs et al., requires the construction of $\mathcal{V}(p)$ for each guard candidate, which in turn requires the processing of many single viewpoint queries for the same terrain. The feasibility of a solution can then be verified with a fast algorithm for multiple viewpoints. Therefore, we are interested in efficient implementations of visibility algorithms, both, for the particular case with a single viewpoint and for the case with multiple viewpoints.

For a single viewpoint we could use existing algorithms for the computation of visibility in polygons by closing the unbounded space above the 1.5D terrain, which transforms it into a simple polygon. This is a well-known problem with a number of established algorithms, see the book by Ghosh [8]. The problem of computing $\mathcal{V}(q)$ was first addressed for simple polygons in [5]. The first correct $O(n)$ time algorithm was given by Joe and Simpson [12]. Regarding algorithms with preprocessing, Ghodsi et al. [10] reduced the query time for simple polygons to $O(\log n + k)$, where k is the complexity of the visibility region, at the expense of preprocessing requiring $O(n^3)$ time and space. In [3] Bose et al. showed that this time can also be achieved for points outside of P with an $O(n^3 \log n)$ preprocessing time, while $O(n^3)$ space remains. Aronov et al. [1] reduced the preprocessing time and space to $O(n^2 \log n)$ and $O(n^2)$ respectively with the query time being increased to $O(\log^2 n + k)$. However, all these algorithms are not applicable because they require too much space. Recently, Bungiu et al. [4] introduced and implemented the *Triangular Expansion* algorithm. The algorithm only requires linear space and, for simple polygons, guarantees $O(n)$ query time. As we reuse some of its ideas, we recap this algorithm in more detail in Section 2.

A naive approach to determine the visibility map for multiple viewpoints is therefore to compute and union the visibility map for each viewpoint, which would take $O(nm)$ time. Recently, Löffler et al. [11, 13] proposed an $O(n + m \log m)$ sweep line algorithm to compute the visibility map for viewpoints placed on the terrain vertices. However, in contrast to the naive approach, the information of $\mathcal{V}(p)$ for each individual viewpoint $p \in P$ is lost. We recap this algorithm as well in Section 2.

1.1 Our Results

- We present a new algorithm – and its implementation – that combines the essential ideas of the Triangular Expansion and the sweep line algorithm of Löffler et al.
- Additionally, we provide an exact and efficient implementation of the sweep line algorithm of Löffler et al., which we extended to handle viewpoints placed arbitrary above the terrain.
- We report on an experimental evaluation of the implementations.
- All implementations will be published as part of the upcoming visibility package of CGAL.

2 Related Algorithms

In this section we review the Triangular Expansion algorithm [4] and the sweep line algorithm for multiple viewpoints in its improved version by Löffler et al. [13]. We incorporate ideas from both algorithms into the new triangular sweep algorithm presented in Section 3.

2.1 Triangular Expansion [4]

The algorithm computes the visibility polygon in $O(nh)$ time, where h is the number of holes of the input polygon P. Thus, for terrains the running time is linear.

The algorithm first triangulates the polygon in a preprocessing step and uses the computed triangulation[1] to achieve better query times. For a query point p, the triangle that contains p is located and from there on, the algorithm proceeds in a recursive manner. It tries to expand the view through every edge of the triangle into the next one. Initially, both endpoints of an edge e restrict the view and the neighboring triangle Δ is entered through e. Then, the view can be restricted further by vertices of P. The algorithm recurses through one or both of the other edges of Δ, depending on whether only one or both of them are seen, respectively.

See Figure 2 for an illustration. Seen from q the vertex v is between ℓ and r, thus both edges e_ℓ and e_r must be considered: e_ℓ is a boundary edge and the algorithm reports edges $\overline{\ell\ell'}$ and $\overline{\ell' v}$; e_r is not a boundary edge, which implies that the recursion continues with v being the vertex that now restricts the left side of the view.

The processing of the queries often run in sublinear time because only the triangles that are seen by the query point p are processed.

[1] We use a constraint Delaunay triangulation.

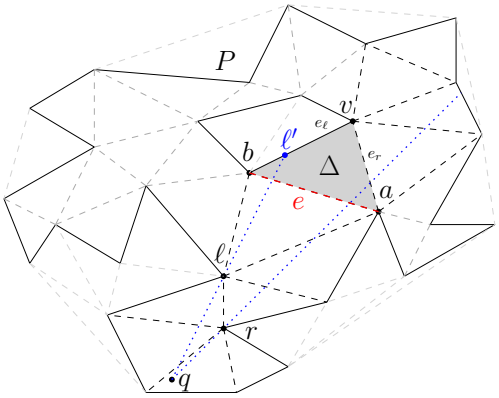


Figure 2: Triangular Expansion algorithm; recursion entering triangle Δ through edge e. [4]

2.2 Sweep by Löffler et al. [13]

The algorithm computes the visibility map in three steps. It sweeps the terrain from left to right and computes the *left-visibility map*. The left-visibility map is defined as all the points on the terrain that are seen from a viewpoint to their left. The *right-visibility map* is analogously defined, and computed with a sweep from right to left. Afterwards both maps are merged. For n vertices and m viewpoints, this takes $O(n+m\log m)$ time. We now discuss the sweep from left to right.

Definition 2.1 *A viewpoint p_1 dominates another viewpoint p_2 at a given x-coordinate x if for all $p \in T$ with $p_x \geq x$ it holds that if p_2 sees p, then p_1 also sees p.*

The following corollary follows from the *order claim* by Ben-Moshe et al. (Claim 2.1 in [2]):

Corollary 2.1 (Corollary 1 in [13]) *Let $q \in T$ be a point visible from viewpoints p_i and p_j, with p_i to the left of p_j. For any $w \in T$ to the right of q, if p_i does not see w, then p_j does not see w either (i.e., p_i dominates p_j at q_x).*

The main idea is to keep track of the changes between visible and not visible while sweeping the terrain, taking advantage of the fact that it is not required to know which points of T are seen by which viewpoint. The status of the sweep line is an ordered set L of shadow rays corresponding to a set of viewpoints that are not visible at the moment; see Figure 3. Every time two rays intersect, only the one with the corresponding viewpoint more to left is relevant for the rest of the algorithm – the other viewpoint is dominated from that point on (see Corollary 2 in [13]). If the terrain is currently seen, in addition, the leftmost viewpoint that currently sees the intersection of the sweep line and the terrain is stored.

The set of all event points is comprised of all the vertices and all the intersections between rays. Dur-

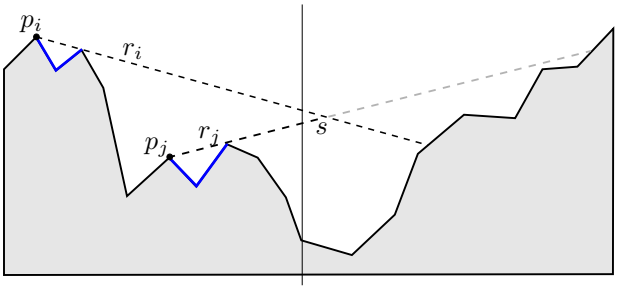


Figure 3: Sweep line algorithm; at the intersection point s, the viewpoint p_j will be dominated by p_i. The corresponding ray r_j can be simply deleted from the status of the sweep line. The left-visibility map w.r.t. the current sweep line is shown in blue.

ing the sweep, the current edge is tested for an intersection with the currently lowermost ray in L to determine when a viewpoint becomes visible again.

3 Triangular Sweep

This algorithm combines the ideas of the two algorithms described in Section 2. It sweeps the terrain essentially the same way as the sweep line algorithm by Löffler et al. does. Specifically, it also performs a sweep for each direction and a subsequent merge step. However, the algorithm only starts with an approximation of the terrain and refines it when necessary.

In a preprocessing step a constraint Delaunay triangulation is computed, with the terrain edges being the constraint edges. This is only done once and only requires $O(n)$ space. A query then starts with a very rough estimation of the terrain, namely the upper part of the convex hull, which can be easily obtained from the triangulation. At first, the algorithm only considers the convex hull edges above the terrain and processes them from left to right (left sweep), i.e., they are kept on a stack with the leftmost edge being the top edge. If a new viewpoint is in between the endpoints of an edge or if a shadow ray intersects the edge, the algorithm refines the terrain and replaces the edge with the two edges below; see Figure 4. Constraint edges are ordinary terrain edges and are handled as such.

With this approach large portions of the terrain that are not seen are skipped, without the need to process every single edge of the terrain. This is worthwhile for $m \ll n$, especially for the particular case when $m = 1$, which is useful when one wants to know the visibility map of each single viewpoint for a set of viewpoints.

The terrain is refined at most $O(n)$ times and each refine step takes only constant time. Thus, the algorithm has the same worst-case running time of $O(n + m \log m)$ as the sweep by Löffler et el.

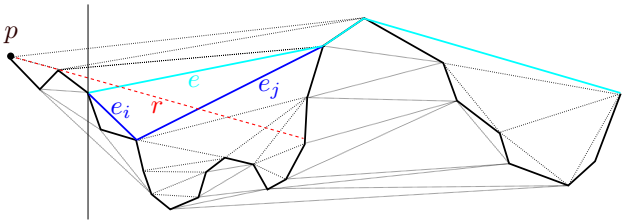


Figure 4: Triangular Sweep; light blue edges are currently on the stack. Edge e is on top and processed next. Ray r intersects edge e, thus e will be replaced by e_i and e_j. Edge e_i will be on top and will be handled next.

4 Experiments

In this section we compare the implementation of the Triangular Sweep with the implementation of the sweep of Löffler et al. [13]. We also compare with the implementations [4] of the Triangular Expansion algorithm and the algorithm by Joe and Simpson. As both algorithms are for polygons we converted the terrain to a simple polygon and only measured the time to compute the visibility polygon, i.e., we did not measure the subsequent conversion to a visibility map. All algorithms are implemented in C++, and are part of the upcoming visibility package of CGAL. As such, all follow the exact geometric computing paradigm.

We have two different scenarios: 1. Single Viewpoints – for some applications the visibility region for each single viewpoint is of interest; see the Integer Program in [6] for example. For this scenario, we used every vertex of a terrain as a separate query. Thus, for a terrain with 1,000 vertices, 1,000 queries were run. 2. Multiple Viewpoints – computing the visibility map for a whole set of viewpoints. That is, we randomly chose a specific percentage of vertices as viewpoints.

We tested two types of terrains, which are depicted in Figure 5 and Figure 6. For the benchmarks we used instances ranging from 100 up to 1,000,000 vertices.

The experiments were run on an Intel Core i5-3210M CPU at 2.5GHz with 3 MB cache and 16 GB main memory running a 64-bit Ubuntu 14.04 LTS operating system. The code was compiled with gcc 4.8.2.

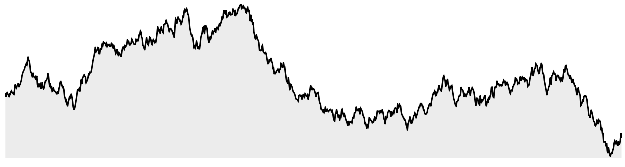


Figure 5: *Random-Walk* instance with 1,000 vertices.

For the first scenario, depicted in Figure 7, we can observe that the preprocessing of the two triangular approaches pays off on larger instances. The difference between the normal sweep and the Triangular Sweep is more than two orders of magnitude

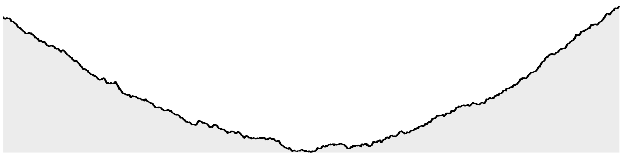


Figure 6: *Parabola-Walk* instance with 1,000 vertices.

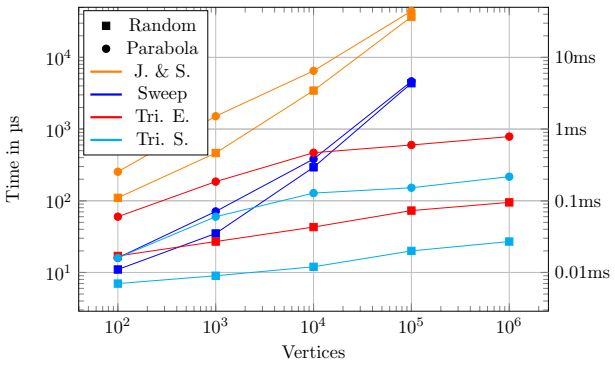


Figure 7: Results for $m = 1$. The average time per query is shown.

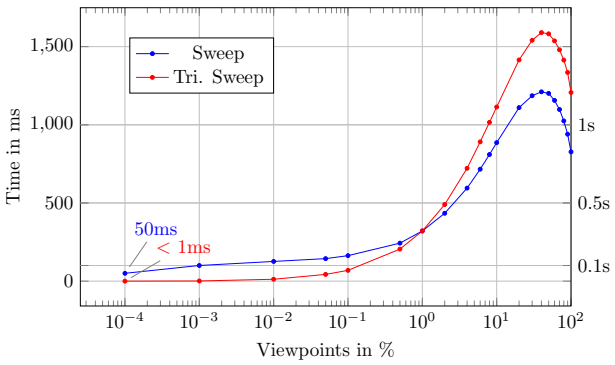


Figure 8: Results for $m \geq 1$ with an instance size of 1,000,000 vertices. A percentage of vertices was chosen at random as viewpoints. The first entry corresponds to exactly one vertex. The results are averaged over 20 different Random-Walk and 20 different Parabola-Walk instances.

for Random-Walk instances of size 10^5 and still more than one order magnitude for the Parabola-Walk instances.

However, for the second scenario (shown in Figure 8) the Triangular Sweep becomes slower at some point. By choosing 1% of the vertices at random as viewpoints, the resulting visibility map already covered roughly 12.5% of the terrain, 10% covered more than 50% and at the peek at 40% viewpoints the visibility map covered around 95% of the terrain. As a result, the additional overhead of maintaining the triangulation edges on the stack outweighed the gain of skipping the unseen terrain parts. Additionally, the cost to maintain the lowermost ray increases with more viewpoints, however, it is not affected at all by the Triangular Sweep.

The drop in runtime at the end is due to two facts; (i) The complexity of the visibility map decreases at some point (we store the projected intervals on the x-axis, i.e., only one interval is stored in case the whole terrain is seen). (ii) The maximum number of simultaneous events in the event queue decreases and drops down to 1 when every vertex is also a viewpoint. Thus, insertions into the queue can be done even without any geometric comparisons at all.

References

[1] B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. Visibility queries and maintenance in simple polygons. *Discrete & Comp. Geometry*, 27(4):461–483, 2002.

[2] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A Constant-Factor Approximation Algorithm for Optimal 1.5D Terrain Guarding. *SIAM J. Comput.*, 36(6):1631–1647, 2007.

[3] P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. *Comput. Geom. Theory Appl.*, 23(3):313–335, 2002.

[4] F. Bungiu, M. Hemmer, J. Hershberger, K. Huang, and A. Kröller. Efficient Computation of Visibility Polygons. *CoRR*, abs/1403.3905, 2014.

[5] L. S. Davis and M. L. Benedikt. Computational models of space: Isovists and isovist fields. *Comput.Gr.Image Process.*, 11(1):49–72, 1979.

[6] S. Friedrichs, M. Hemmer, and C. Schmidt. A PTAS for the continuous 1.5D Terrain Guarding Problem. *CoRR*, abs/1405.6564, 2014.

[7] S. Friedrichs, M. Hemmer, and C. Schmidt. Exact solutions for the continuous Terrain Guarding Problem. In *EuroCG*, 2015 (submitted).

[8] S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 2007. Cambridge Books Online.

[9] M. Gibson, G. Kanade, E. Krohn, and K. Varadarajan. An Approximation Scheme for Terrain Guarding. In *Approx., Random., and Comb. Optim. Algorithm. and Tech.*, volume 5687 of *LNCS*, pages 140–148. 2009.

[10] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. *SIAM J. Comput.*, 26(4):1120–1138, 1997.

[11] F. Hurtado, M. Löffler, I. Matos, V. Sacristan, M. Saumell, R. I. Silveira, and F. Staals. Terrain Visibility with Multiple Viewpoints. In L. Cai, S.-W. Cheng, and T. W. Lam, editors, *ISAAC*, volume 8283 of *LNCS*, pages 317–327. Springer, 2013.

[12] B. Joe and R. B. Simpson. Corrections to Lee's Visibility Polygon Algorithm. *BIT*, 27(4):458–473, 1987.

[13] M. Löffler, M. Saumell, and R. I. Silveira. A faster algorithm to compute the visibility map of a 1.5D terrain. In *EuroCG*, Mar.

Experiments on Parallel Polygon Triangulation Using Ear Clipping

Günther Eder* Martin Held* Peter Palfrader*

Abstract

We present an experimental study of different strategies for triangulating polygons in parallel. As usual, we call three consecutive vertices of a polygon an ear if the triangle that is spanned by them is completely inside of the polygon. Extensive tests on thousands of sample polygons indicate that most polygons have a linear number of ears. This experimental result suggests that polygon-triangulation algorithms based on ear clipping might be well-suited for parallelization.

We discuss three different on-core approaches to parallelizing ear clipping and report on our experimental findings. Extensive tests show that the most promising method achieves a speedup by a factor of roughly k on a machine with k cores.

1 Introduction

An ear of a planar simple polygon P is formed by three consecutive vertices (v_{i-1}, v_i, v_{i+1}) if the open line segment $\overline{v_{i-1}, v_{i+1}}$ is completely contained in the interior of P (see Fig. 1). It is well-known [2] that (v_{i-1}, v_i, v_{i+1}) form an ear of P if and only if (i) v_i is convex, and (ii) the closure of the triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ does not contain any reflex vertex of P (except possibly v_{i-1} or v_{i+1}). Hence, if (v_{i-1}, v_i, v_{i+1}) is an ear of P then the line segment $\overline{v_{i-1}, v_{i+1}}$ forms a diagonal of P. Clipping this ear by inserting this diagonal cuts off the vertex v_i, the "base" of the ear, thus reducing the number of vertices of P by one.

The basic idea of ear clipping is to iteratively cut off ears until the polygon has shrunk to a triangle. The algorithm's correctness hinges upon Meisters' two-ears theorem which states that every simple polygon with four or more vertices has at least two non-overlapping ears [4].

Typically, an implementation of an ear-clipping algorithm will operate in two phases. *Classification:* Iterate along the contour of P to determine all instances of three consecutive vertices that form an ear of P. All potential ears are stored in a queue. *Clipping:* Iteratively pick an ear from the queue and clip it. As an ear (v_i, v_j, v_k) is clipped and stored in a triangle list, its two outer vertices v_i and v_k have to be checked whether they form the bases of new ears after the clipping of (v_i, v_j, v_k). Every newly found ear is added to the queue. Note that the queue may contain candidate ears which are no longer part of the polygon. The process ends for an n-vertex input polygon P when $n-3$ ears have been clipped and, thus, the triangle list together with the final triangle forms a complete triangulation of P.

Ear clipping forms the basis of the FIST triangulation algorithm and ANSI-C implementation, Held's fast industrial-strength triangulation tool [2]. Key features of FIST include speed and robustness. While the basic ear-clipping algorithm has an $\mathcal{O}(n^2)$ worst-case complexity, FIST employs multi-level geometric hashing to speed up the computation to near-linear time for almost all (real-world and contrived) inputs. Extensive tests [3] showed that FIST's careful engineering allows it to run flawlessly on a standard floating-point arithmetic.

Ear clipping is, ostensibly, limited to triangulating simple polygons. FIST, however, also handles polygons with holes by converting them in a preprocessing step: so-called *bridges* are inserted to connect all hole polygons directly or indirectly to the outer boundary polygon, turning a polygon with holes into one (slightly degenerate) simple polygon, which can then be triangulated using ear clipping.

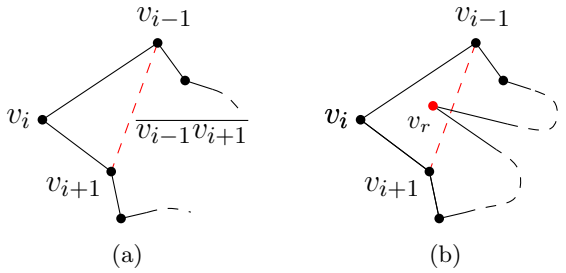


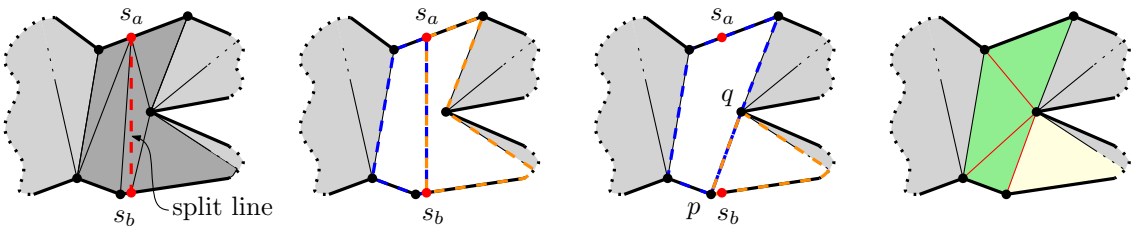
Figure 1: (a) An ear defined by the vertices v_{i-1}, v_i, v_{i+1} where v_i is convex; (b) A reflex vertex v_r violates the second condition.

2 Prior Work

Surprisingly little work has been done on parallel triangulations. The literature focuses mostly on (Delaunay) triangulations of point sets rather than polygons, see for instance Rong et al. [6] and Xin et al. [8].

In 2013, Qi et al. [5] introduced a primarily GPU-based algorithm to compute constrained Delaunay tri-

*Universität Salzburg, FB Computerwissenschaften, 5020 Salzburg, Austria; supported by Austrian Science Fund (FWF) Grant P25816-N15; {geder,held, palfrader}@cosy.sbg.ac.at.



(a) (b) (c) (d)

Figure 2: The repair process used in the divide and conquer algorithm.

angulations. In a first step they compute a Voronoi diagram, i.e., the dual of the Delaunay triangulation. Constraints are then added to obtain the constrained Delaunay triangulation (CDT). Their approach scales well on the GPU and seems to be the currently best solution if an NVIDIA GPU is available.

3 Our Contribution

We study the prevalence of ears in our vast set of test data (see Sec. 5) and find that, on average, about half of all vertices of a given polygon form the bases of ears. If we look only at convex vertices then a vast majority (98 %) of them belong to ears. This is significantly more than the two ears guaranteed by Meisters' theorem [4] and, hence, suggests that clipping many ears simultaneously is feasible.

We therefore extend the classic FIST ear-clipping algorithm such that it can operate in parallel. We present three particular variants: a divide-and-conquer algorithm, an algorithm that uses a partitioning of the contour and a mark-and-cut approach. All algorithms were implemented within the FIST framework and compared to the conventional FIST.

4 Parallel Ear-Clipping Algorithms

In the sequential version of FIST, on average, about 80 % of the time is spent for classification and clipping of the ears, while only approximately 20 % is spent on preprocessing, such as data cleaning and bridge finding to convert polygons with holes into degenerate (weakly-)simple polygons. We therefore concentrate our parallelization efforts on the classification and clipping phases.

4.1 Divide and Conquer

The basic idea is to split the polygon into as many sub-polygons as CPU cores are available. All sub-polygons shall have roughly the same number of vertices. Since it seems costly to determine suitable diagonals that achieve balanced splits, we simply use vertical lines to split the polygon. Using the Sutherland-Hodgman algorithm [7], we can split a polygon along a line ℓ in time $\mathcal{O}(n)$, at a cost of at most $\mathcal{O}(n)$ Steiner points given by the number of intersections between ℓ and the edges of the polygon. (In practice, the number of Steiner points seems to be bounded by $\sqrt{n}$ for almost all but contrived inputs.)

We then run one (sequential) FIST instance per core to obtain a triangulation of each sub-polygon. A concatenation of the triangulations of all sub-polygons yields a triangulation of P, albeit with Steiner points which have to be removed.

Consider a pair of consecutive Steiner points s_a and s_b. We remove them and all their incident triangles, and we repair the contour by re-joining vertices that were adjacent previously. The removal of incident triangles leaves a hole H which forms a "double star-shaped" polygon, where every point of H is visible from at least one of the Steiner points s_a and s_b, see Fig. 2b.

Using s_a as start vertex, we walk counter-clockwise along the boundary of H. If we find a vertex that is not visible from s_a, then we store the preceding vertex as p. If all vertices are visible, then we stop the test when we reach the second Steiner point s_b and store the last vertex before s_b as p. Then we start the same search clockwise, starting again at s_a, and obtain q.

We divide H into two star-shaped polygons H_1, H_2 by adding the diagonal $\overline{pq}$. Each of them has one of the original Steiner points in its nucleus, see Fig. 2c. Now every triangle based at a convex vertex of H_1 and H_2 forms an ear as long as it does not contain either s_a or s_b. Hence, both holes can be triangulated easily.

4.2 Partition and Cut

In this approach, we use the sequential classification step to partition the contour of the polygon into k different sets. We choose k to correspond to the number of cores we want to run on. The sequential FIST walks along the polygon, tests each vertex triple for its ear property, and stores all ears in one queue. To parallelize, we set the number of queues equal to k and split the polygon into k polygonal chains with roughly n/k many vertices each. Additionally, we memorize k contour vertices between the chains. These *corner vertices* are important for the parallel clipping to ensure that no thread oversteps its partition boundaries;

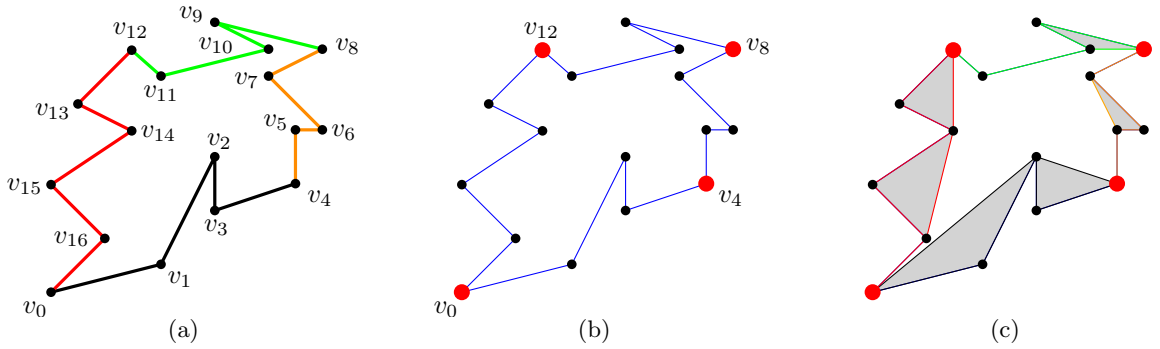


Figure 3: (a) A simple polygon P already partitioned into four chains. (b) The four corner vertices are highlighted. (c) A partition-and-cut triangulation.

see Fig. 3b.

In the parallel clipping phase, each thread processes all ears from its queue. As usual, clipping the ear (v_{i-1}, v_i, v_{i+1}) involves checking whether $v_{i\pm1}$ has become the basis of a new ear. If so, and if $v_{i\pm1}$ is no corner vertex, then $v_{i\pm1}$ is added to the queue.

After all k queues are empty, the parallel clipping phase is completed. Now we finish the triangulation with a sequential run of FIST. This will remove the remaining vertices from our polygon.

4.3 Mark and Cut

This algorithm builds on the fact that every second ear along the polygon's contour is non-overlapping. Hence, we can mark those ears and clip them immediately without conflicts. We only need one additional data structure, namely an array A to store the indices of all marked ears.

In the *mark phase*, we walk along the polygon once and store the index of every other vertex in A. Additionally we only take convex vertices and add flags to check if a triangle has already been checked for its ear-property. In the *cut phase*, we check for each vertex v_i in A whether (v_{i-1}, v_i, v_{i+1}) forms an ear. If so, we clip this ear and store the triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ in the triangle array at position i. Since we considered only every other vertex, ears that we find cannot overlap.

Every ear can be clipped only once and for every clipped ear only one vertex is removed from the polygon. Thus, we can use the index i of a removed vertex v_i as an index for the stored triangle and avoid any collisions.

The algorithm is designed to work in parallel without locks or atomics. Initially, we start the mark phase and mark the first half of the contour as described above. Then we let all threads but one run through the first half of the array A in parallel and check each vertex (cut phase). The remaining thread marks the second half of the contour and stores the vertex indices in the second half of A. After all threads are finished, the procedure starts again with marking the first half of the remaining contour and cutting the vertices stored in the second half of A and so on.

Once only a low number of new triangles can be generated in a cut phase, we run the sequential version of FIST on the remaining polygon.

5 Experimental Results

We implemented the parallel variants of FIST as an on-core parallelization by the use of OpenMP/C++.

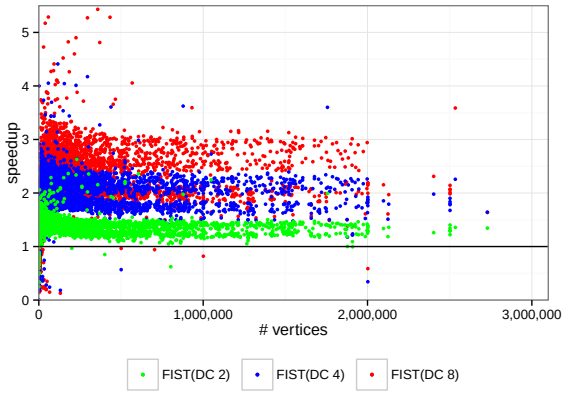
Our test system runs CentOS 6.5 on an 2014 Intel Xeon E5-2667 v3 CPU at 3.20 GHz with 8 cores and 132 GB RAM.

Our implementations were tested on about twenty thousand polygons with up to four million vertices per input, consisting of both real-world and contrived data of different characteristics, including CAD/CAM designs, printed-circuit board layouts, geographic maps, closed fractal and space filling curves, star-shaped and random polygons generated by RPG [1], as well as sampled spline curves and font outlines. Some datasets contain circular arcs, which we approximated by polygonal chains in a preprocessing step. Similarly, we used the standard (sequential) FIST to convert all multiply-connected polygonal areas to (slightly degenerate) simple polygons by inserting bridges.

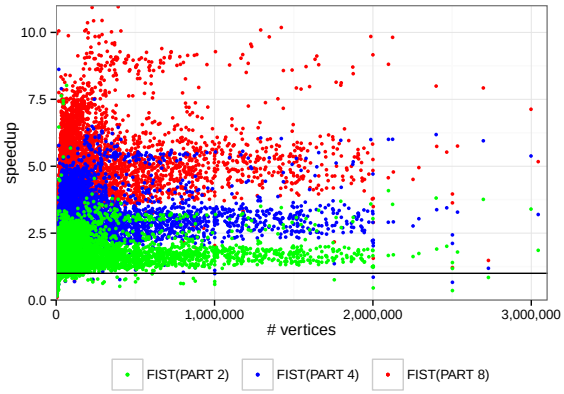
In our tests, we compare the runtime of our parallel algorithms to the runtime of the conventional sequential FIST. The plots of Fig. 4 show the speedups that we achieved.

We observe the overall best result for the mark-and-cut algorithm (Fig. 4c), with an average speedup of 8 when using eight cores or 4 when using four cores. In any case, parallelization seems to pay off once the input polygon has at least about fifteen thousand vertices. Some test sets yield a speedup of over k while employing k cores. This is a result of the different storage structure used in the sequential version of FIST.

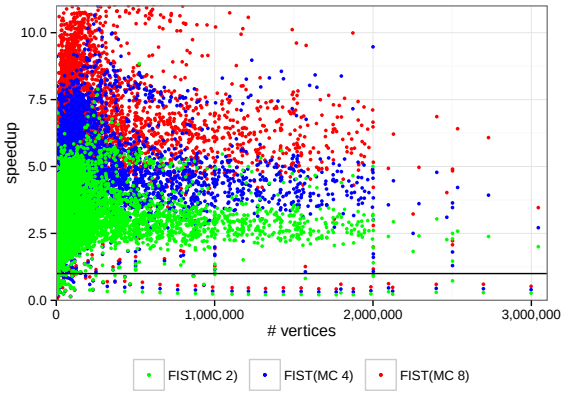
Tests on other systems with up to 64 cores did not



(a) Divide and Conquer



(b) Partition and Cut



(c) Mark and Cut

Figure 4: Speedup of parallelization approaches as a function of input size. In (a), (b), and (c) we see speedups for different numbers of threads. For the divide-and-conquer approach this is the same as the number of sub-polygons used.

scale as well as presumed. However, those systems were multi-CPU systems which seem to behave differently than the multi-core system which we used, for reasons not yet fully understood.

Summarizing, we present experimental evidence that an algorithm for triangulating polygons based on ear clipping can be parallelized for efficient execution on multi-core computers. Since current personal computers are equipped with quad-core processors, the triangulation of a polygon can be accomplished about four times as fast with our parallel variants of FIST in most cases.

References

[1] T. Auer and M. Held. Heuristics for the Generation of Random Polygons. In *Proc. 8th Canad. Conf. Comput. Geom (CCCG 1996)*, pages 38–44, Aug. 1996.

[2] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica*, 30(4):563–596, 2001.

[3] M. Held and W. Mann. An Experimental Analysis of Floating-Point Versus Exact Arithmetic. In *Proc. 23rd Canad. Conf. Comput. Geom. (CCCG 2011)*, pages 489–494, Aug. 2011.

[4] G. H. Meisters. Polygons have Ears. *The American Mathematical Monthly*, 82(6):648–651, June 1975.

[5] M. Qi, T. Cao, and T. Tan. Computing 2D Constrained Delaunay Triangulation Using the GPU. *IEEE Trans. Visualizat. Comput. Graph.*, 19(5):736–748, May 2013.

[6] G. Rong, T.-S. Tan, T.-T. Cao, and Stephanus. Computing Two-Dimensional Delaunay Triangulation using Graphics Hardware. In *Proc. ACM Symp. Interactive 3D Graphics*, I3D '08, pages 89–97, 2008.

[7] I. E. Sutherland and G. W. Hodgman. Reentrant Polygon Clipping. *C. ACM*, 17(1):32–42, Jan. 1974.

[8] S.-Q. Xin, X. Wang, J. Xia, W. Mueller-Wittig, G.-J. Wang, and Y. He. Parallel Computing 2D Voronoi Diagrams using Untransformed Sweepcircles. *Computer-Aided Design*, 45(2):483–493, 2013.

Kinetic Conflict-Free Coloring*

Mark de Berg† Tim Leijssen† Marcel Roeloffzen‡

Abstract

A conflict-free coloring, or CF-coloring for short, of a set P of points in the plane with respect to disks is a coloring of the points of P with the following property: for any disk D containing at least one point of P there is a point $p \in P \cap D$ so that no other point $q \in P \cap D$ has the same color as p. In this paper we study the problem of maintaining such a CF-coloring when the points in P move. We present two methods for this and evaluate the maximum number of colors used as well as the number of recolorings, both in theory and experimentally.

1 Introduction

Wireless communication is commonplace in many applications, varying from mobile consumer devices like cell-phones to specialized networks for autonomous robots. For clear wireless communication it is important to avoid interference. In general interference-free communication is achieved by using different frequencies for different communication channels. However, the number of available frequency ranges is limited so it is impossible to assign a unique frequency to every communication channel. A more clever frequency-assignment scheme is therefore needed. Finding a proper assignment of frequencies is often modeled as the problem of computing a so-called *conflict-free coloring*, or *CF-coloring* for short, where different colors represent different frequencies.

In this paper we look at the problem of finding a CF-coloring with respect to disks for a set P of points in the plane. That is, we want to assign colors to the points of P so that every disk D containing at least one point of P contains a unique color. More formally, we want to assign a color $color(p)$ to each point $p \in P$ so that the following property holds: for every disk D containing at least one point of P, there is a point $p \in P \cap D$ so that for any point $q \in P \cap D \backslash \{p\}$ we have $color(p) \neq color(q)$. It is convenient to identify the i-th color with the integer i. The goal is then to minimize the maximum color used in the CF-coloring.

The problem was introduced by Even *et al.* [5] who provide a framework for finding CF-colorings with respect to disks and several other types of regions. Their algorithm works by finding independent sets on a sequence of Delaunay triangulations of subsets of P and uses $O(\log n)$ colors, which is asymptotically optimal [5, 7]. Several variants of the problem have also been studied. For example, Har-Peled *et al.* [6] provide a probabilistic algorithm that works for simple regions with low union-complexity, as well as several algorithms for special cases such as axis-aligned rectangles. They also study k-CF-coloring, which is a relaxation of a regular CF-coloring. In a k-CF-coloring each region need not contain a unique color, but instead a color that occurs between 1 and k times in that region. There are also several results for online CF-coloring, where points are added incrementally and must be assigned a color when added without knowledge of points to come [2, 4]. Additional background can be found in a recent survey by Smorodinsky [8].

All these papers assume the transmitters are stationary and frequencies are assigned exactly once. This may however not always be the case. For mobile RFID scanners or networks of autonomous robots, for example, the frequency assignment may need to change as the transmitters move. Unfortunately changing the frequency of a transmitter is often undesirable as it may interrupt current communications with that transmitter. This leads us to study the problem maintaining a conflict-free coloring in the so called *Kinetic Data Structures (KDS)* framework.

Following the KDS framework as introduced by Basch *et al.* [3] we assume that we know the trajectories of the points (at least in the short term). The goal is now to maintain a CF-coloring of the points as they move along these trajectories. In frequency assignment, changing the frequency of a transmitter is usually expensive. Therefore, we focus on keeping the number of recolorings low while using $O(\log n)$ colors. We provide two algorithms for this, both based on the algorithm by Even *et al.* [5]. For both algorithms we provide a theoretical analysis and an experimental analysis of the maximum number of colors used and of the number of recolorings.

2 The algorithms

First we describe the algorithm by Even *et al.* [5] for static points in more detail, as our kinetic algo-

*MdB is supported by the Netherlands Organisation for Scientific Research (NWO) and MR is supported by the SMART project at Tohoku University.

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

‡Graduate School of Information Sciences, Tohoku University, Japan. *marcel@dais.is.tohoku.ac.jp*

rithms are based on this. The algorithm is recursive and starts by computing the Delaunay triangulation $\mathcal{DT}(P_1)$ of the complete point set $P_1 := P$. The next step is to find a large independent set $I_1 \subset P_1$ in $\mathcal{DT}(P_1)$. All points in I_1 receive color 1. On the set $P_2 := P_1 \backslash I_1$ a new Delaunay triangulation and independent set I_2 are computed, then the points in I_2 receive color 2, and the algorithm proceeds with $P_3 := P_2 \backslash I_2$. This process is repeated until all points are colored. If we compute the independent set in each $\mathcal{DT}(P_i)$ in a greedy manner, by considering the points in order of increasing degree, we can guarantee that $|I_i| \geqslant |P_i|/6$, which implies that the algorithm finishes after $O(\log n)$ rounds. The proof that this procedure produces a conflict-free coloring was given—in a more general setting—by Even *et al.* [5].

Next we adapt the static algorithm to a kinetic setting where points move along known constant-complexity curves. From the above it is clear that as long as none of the Delaunay triangulations change, then the coloring remains conflict-free. However, as the points move the Delaunay triangulation can change, namely when four points that form a quadrangle in the Delaunay triangulation become co-circular. When this happens a diagonal of the quadrangle flips, which may invalidate the independent set. Under known motions these co-circularities are not difficult to detect and kinetic data structures exist that maintain a Delaunay triangulation and support insertion and deletion of vertices [1]. Therefore we focus on repairing the independent sets when such a flip occurs. We propose two different methods, a greedy method and a lazy method.

In the following we describe what the two methods do when a flip occurs between vertices within a quadrangle $Q := Q(p, q, r, s)$, where the edge pr is replaced by qs. The quadrangle Q may exist in the Delaunay triangulation of multiple consecutive levels, but only in the last level i where it occurs can one or more of the points p, q, r, s be in the independent set. This follows from the fact that a point in the independent set of level i cannot occur in any level $j > i$. We denote this level by $lev(Q)$ and use k to denote the total number of colors used before recovering.

Lazy updates. In the lazy approach we check if the new edge connects two points in the independent set, thus making the set no longer independent. If so, we recolor one of the two conflicting vertices to a new color. As a result the independent set of each layer will slowly become smaller compared to the total number of points in that layer. When the independent set becomes too small, we completely recompute the structure from that layer onward.

In more detail, let $Q := Q(p, q, r, s)$ denote the quadrangle within which the flip occurred, where edge pr is replaced by qs. Let $i := lev(Q)$. When the edge pr flips to edge qs, we check if q and s are both in the independent set I_i. If so, we remove one of the two, say q, from I_i, give it a new color $k+1$ and add it to all sets P_j for $i < j \leqslant k+1$. Note that any two points that were independent in $\mathcal{DT}(P_j)$ are still independent in $\mathcal{DT}(P_j \cup \{q\})$. Since we reduced I_i in size we check if it has become too small. More precisely, we check if $|I_i| < |P_i|/12$, and if so we recolor P_i entirely by running the static algorithm on P_i. We call this a *reset* at level i. If we do not do a reset at level i, we check if $|I_j| < |P_j|/12$ for some $j > i$. (This is needed since we added q to all sets P_j for $i < j \leqslant k+1$.) If this is the case for some levels j, we do a reset at the smallest such level j.

Lemma 1 *Maintaining a CF-coloring with lazy updates ensures that we use at most $O(\log n)$ colors at any time.*

Proof. Each update guarantees that the sets I_i are independent. Following the proof of Even *et al.* [5] this is enough to enough to show the coloring remain conflict-free. Since we guarantee that each independent set I_i contains at least $|P_i|/12$ points at any time, $O(\log n)$ colors are used. □

Our goal is to guarantee a CF-coloring using as few recolorings as possible. Therefore, we now study the number of recolorings that this method makes. It is easy to see that a single flip in a Delaunay triangulation may cause $\Theta(n)$ recolorings: if a flip triggers a reset of the first level, then all points may be recolored. However, this does not happen often since our greedy independent-set computation guarantees that initially $|I_i| \geqslant |P_i|/6$, whereas we won't reset until $|I_i| < |P_i|/12$. Most events—that is, flips in the Delaunay triangulation—do not cause many recolorings and we can in fact prove that amortized only $O(\log n)$ recolorings happen per event.

Lemma 2 *Each event triggers $O(\log n)$ recolorings amortized.*

Proof. We prove this using an accounting scheme, where each point p has a wallet $W_i(p)$ with a certain amount of money, for every level i where it occurs. To recolor a point it must pay €1. Thus, when doing a reset at level i, all points in P_i must pay €1. Next we show that in each event we have to spend only $O(\log n)$ euro to guarantee we can pay for all recolorings. We keep as an invariant that any point p in every level i where it occurs has at least

$$1 - \frac{|I_i| - |P_i|/12}{|P_i|/12} = 2 - \frac{12|I_i|}{|P_i|} \text{ euros}$$

in its wallet $W_i(p)$. Note that since $|I_i| \geqslant |P_i|/6$ after a reset at level i, the points need not have any money

immediately after a reset. Moreover, a reset occurs when $|I_i| < |P_i|/12$, so then each point has at least €1 and can pay for its recoloring.

Next we show we need to spend no more than $O(\log n)$ euro per event to maintain the invariant. Consider an event that causes point q to be removed from the independent set I_i and added to a new level P_{k+1}. The wallets for points in levels $j < i$ do not change as the set P_j and I_j do not change. In level i the size of P_i remains the same, but I_i becomes one smaller. Let I_i and P_i denote the independent set and complete set of points in level i before the event and I_i' and P_i' the same sets after the event. Before the event we know that each point $p \in P_i$ has at least $2 - 12|I_i|/|P_i|$ euro. After the event it should have at least $2 - 12|I_i'|/|P_i'| = 2 - 12(|I_i| - 1)/|P_i|$ euro. To ensure this we must give each point at most $12/|P_i|$ euro, so at most €12 in total for all points in level i.

The money to be payed to levels $j > i$ is calculated as follows. Before the event each point in level j has at least $2-12|I_j|/|P_j|$ euro and after the event it must have at least $2 - 12|I_j|/(|P_j| + 1)$ euro. This means we should pay each point in P_j (not including q)

$$\frac{12|I_j|}{|P_j|} - \frac{12|I_j|}{|P_j|+1} = \frac{12|I_j|}{|P_j|(|P_j|+1)} \leqslant \frac{12}{|P_j|+1} \text{ euro.}$$

Since we have to pay this to $|P_j|$ points, this costs us no more than €12 per level. Lastly we have not yet filled the wallet for the point q in all the levels it has been added to. However, in each level it requires at most €1. In total we spend no more than €13 per level, so $O(\log n)$ euro in total. □

Greedy updates. In the lazy approach we do $O(\log n)$ recolorings amortized per event, but in some updates we may have to recolor all points. We can try to avoid this worst-case behavior by keeping the independent sets large. To this end, at each event we not only remove a point from an independent set, but we also try to add new points. We maintain as an invariant that each independent set is a maximal independent set consisting only of points with degree less than 12. This produces an independent set I_i with $|I_i| \geqslant |P_i|/24$, as at least half the vertices have degree less than 12 and choosing one of these eliminates no more than 12 from becoming part of the independent set. Next we discuss the updates necessary to maintain this invariant.

Consider an event that is a flip within quadrangle $Q := Q(p, q, r, s)$, where the edge pr is replaced by edge qs. We denote by $lev^*(Q)$ the first level (that is, with smallest index i) where this flip occurs. When processing the event we start at $lev^*(Q)$ and then proceed to later levels. In each level we have the following situation. We have a current set P_i with Delaunay triangulation $\mathcal{DT}(P_i)$ and two (possibly empty) sets P_i^+ and P_i^-. The set P_i^+ contains points that are "pushed down" from the parent level because they are no longer in the independent set at that level; the set P_i^- contains points that are "pulled up" from the parent level because they have just been added to the independent set at the parent level. We then update the Delaunay triangulation by constructing $\mathcal{DT}((P_i \cup P_i^+) \setminus P_i^-)$, update the independent set I_i, and construct the sets P_{i+1}^+ and P_{i+1}^- by looking at the differences between old and new independent set I_i.

In more detail we proceed as follows. We start by computing $\mathcal{DT}((P_i \cup P_i^+) \setminus P_i^-)$. Note that for $i = lev^*(Q)$ we have $P_i^+ = P_i^- = \emptyset$, but the Delaunay triangulation still changes because of the flip in Q. Let P_i^* denote the set of points whose neighbor set has changed, including the set P_i^+ of new points. Now for the points in $P_i^* \cap I_i$ we test if they still have degree below 12, otherwise we remove them from I_i. Then we check if any two vertices remaining in the independent set are connected, and we remove one of them if needed. We repeat this until no more connected pairs exist, and we have obtained an independent set for the new Delaunay triangulation. Next we make the independent set maximal by greedily adding points with degree less than 12 that are not connected to any points of the independent set. The resulting set is the new independent set I_i. As already mentioned, we then construct the sets P_{i+1}^+ and P_{i+1}^- by looking at the differences between old and new independent set I_i. Note that the sets P_i^+ and P_i^- (and, hence, the set P_i^* of "affected points") may grow as we go down the levels. Unfortunately it seems hard to prevent this, which makes it difficult to bound the worst-case number of recolorings. Fortunately our experiments (see below) show that in practice the avalanche effect is limited and not too many recolorings occur per event.

3 Experimental results

We showed some basic theoretical bounds for the number of recolorings. However, especially for the greedy approach it seems unlikely that in practice we need many recolorings per step. We therefore implemented both algorithms in order to count the number of recolorings. For our input we use sets of points that move along straight lines within a bounding box. To avoid points going outside the bounding box, they bounce back when hitting the edge of bounding box. We ran our implementation of the two methods for sets of moving points ranging from 20 to 2000 points and measure the number of recolorings per step as well as the total number of colors used. For 100 points the results can be found in Fig. 1. The results with other amounts of points are summarized in Table 1 and are based on a run of 10,000 events—that is, flips in any of the Delaunay graphs.

The graphs of Fig. 1 clearly show the difference between the two methods. The lazy method has a very

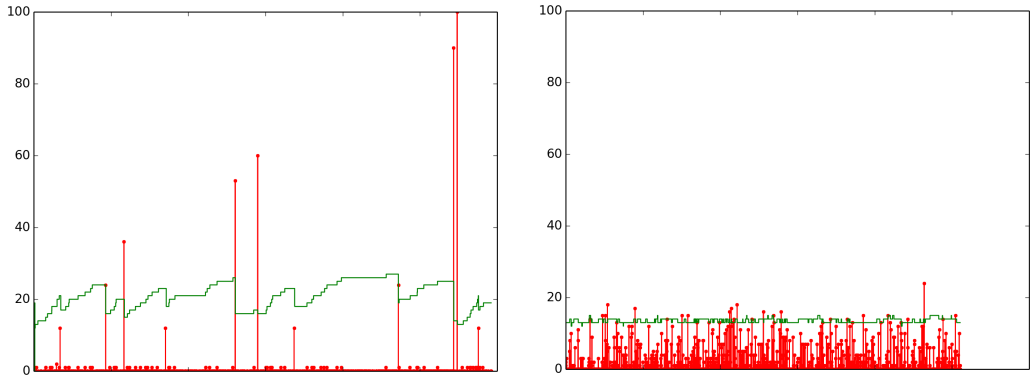


Figure 1: Number of recolorings indicated in red and total number of colors used in green for the lazy (left) and greedy (right) method. Both are measured from an instance with 100 moving points over 2500 events.

	Lazy Updates						Greedy Updates					
	#Colors				#Recolorings		#Colors				#Recolorings	
n	avg	max	$\frac{\text{avg}}{\log n}$	$\frac{\text{max}}{\log n}$	total	max	avg	max	$\frac{\text{avg}}{\log n}$	$\frac{\text{max}}{\log n}$	total	max
20	12.4	15	2.9	3.5	1232	20	8.2	10	1.9	2.3	6414	12
50	17.2	22	3.0	3.8	1569	50	11.2	14	2.0	2.5	8036	19
100	20.9	28	3.1	4.2	2017	100	13.5	16	2.0	2.4	9834	24
200	24.6	31	3.2	4.1	2275	200	15.9	18	2.1	2.4	12034	27
500	28.3	35	3.2	3.9	2879	337	19.1	21	2.1	2.3	15248	34
1000	31.3	38	3.1	3.8	4064	444	21.3	23	2.1	2.3	18406	62
2000	34.0	42	3.1	3.8	5659	696	23.5	25	2.1	2.3	22164	60

Table 1: Number of colors used and recolorings for n moving points using lazy and greedy updates.

low number of recolorings for most events, but several events with a many recolorings that correspond to resets at top levels of our data structure. This step-wise behavior is also found in the total number of colors used, which slowly goes up until a reset of one of the top levels is done. The greedy approach on the other hand does not have this spiky behavior, and instead many events cause between 5 and 20 recolorings, but rarely more than that. It also has the effect that the total number of colors appears very stable.

In Table 1 we see that in both cases the number of colors used is proportional to $\log n$. What is interesting though is that although the greedy method uses less colors, both on average and maximum, it requires many more recolorings on average. This shows that the neither of the two methods is strictly better than the other. If one can afford to do many recolorings every now and then, and one does not care about using a few more colors, then the lazy method is better as it will do fewer recolorings. On the other hand, if the goal is to keep the total number of colors very small or to avoid many recolorings at a single event, then the greedy method appears better.

References

[1] G. Albers, L.J. Guibas, J.S.B. Mitchell, and T. Roos. Voronoi diagrams of moving points. *Int. J. Comput. Geometry Appl.* 8(3):365–380, 1998.

[2] A. Bar-Noy, P. Cheilaris, S. Olonetsky, and S. Smorodinsky. Online conflict-free colorings for hypergraphs. *Journal Combinatorics, Probability and Computing* 19(4):493–526, 2010.

[3] J. Basch, L.J. Guibasand, J. Hershberger. Datastructures for mobile data. In *Proc. 8th ACM-SIAM Symp. Disc. Alg.*, pages 747–756, 1997.

[4] K. Chen, A. Fiat, H. Kaplan, M. Levy, J. Matoušek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, and U. Wagner. Online conflict-free coloring for intervals. *SIAM Journal on Computing* 36(5):1342–1359, 2006.

[5] G. Even, Z. Lotker, D. Ron, S. Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM Journal on Computing* 33(1):94–136, 2003.

[6] S. Har-Peled and S. Smorodinsky. Conflict-free coloring of points and simple regions in the plane. *Discrete & Computational Geometry* 34(1):47–70, 2005.

[7] J. Pach and G. Toth. Conflict-free colorings. In *Discrete and Computational Geometry - The Goodman-Pollack Festschrift* (S. Basu et al., eds.), Springer Verlag, Berlin, pages 665–671, 2003.

[8] S. Smorodinsky. Conflict-Free Coloring and its Applications. In *Geometry - Intuitive, Discrete, and Convex* (I. Bárány et al., eds.) Springer Verlag, Berlin, 2014.

Kinetic Data Structures for Clipped Voronoi Computations

Duru Türkoğlu*

Abstract

We consider the mesh refinement problem in the kinetic setting: given an input set of moving points, the objective is to design a kinetic data structure (KDS) for inserting additional so-called Steiner points so that the resulting output set yields a quality triangulation. Therefore, the selection of Steiner points plays a cruicial role, both in terms of the output itself and the quality of the triangulation. Although many Steiner point selection schemes have been devised, it is not straightforward how to adapt them in the kinetic setting; it may not even be possible to adapt some of these schemes.

In this paper, we design a KDS by extending a previously proposed query structure which has been employed for Steiner point selection in the dynamic setting. The key geometric property computed in these structures is the clipped Voronoi cells, a locally restricted version of the standard Voroni cells. Our KDS maintains these clipped Voronoi cells, where each query takes constant time to compute as well as to update. Hence, our KDS is responsive, and it is efficient processing a constant number of events for each query, and it is also local and compact.

1 Introduction

Mesh refinement is a fundamental step in scientific computing, where the goal is to construct a quality triangulation of a given set of input points. For most applications, a quality triangulation is one in which the minimum angle of every triangle is above some threshold. Until this quality criterion is satisfied, one needs to *refine* the trianguation by inserting additional *Steiner* points into the output, taking care to insert as few as possible. If the minimal output that admits a quality triangulation has m points, any output of size $O(m)$ is regarded as *size-optimal*.

Over the past twenty years, research has provided improved mesh refinement algorithms with theoretical guarantees. In chronological order, these guarantees were quality [4], size-optimality [8], efficient runtimes [5, 6], and efficient dynamic updates [1], each of them incorporating all of the previous guarantees. And most recently, Acar et al. developed a KDS for mesh refinement [2]. Their choice of Steiner points, however, only allowed bounded quality triangulations. Hence, mesh refinement problem in the kinetic setting still remains open for higher quality triangulations.

In the kinetic setting, the objective of mesh refinement is to maintain a set of moving Steiner points as well as the triangulation of the output set comprising input and Steiner points. Typically, algorithms used in practice maintain the moving mesh by a series of further refinements to fix low quality elements over time, and by remeshing occasionally to bound the size of the mesh. In this paper, we approach the problem using the kinetic data structures (KDS) framework introduced by Basch et al [3]. In this framework, the input points are provided with algebraic trajectories of constant degree and the computed geometric property is certified by a set of *certificates*, some of which may fail at a later time causing an *event*. Each event triggers a kinetic update for repairing both the geometric property itself and the set of certificates certifying it. As the trajectories of the input points are known in advance, the data structure maintains the desired geometric properties by processing these events in order of their failure times. In this framework effectiveness is measured by the following criteria: a KDS is called *responsive* if the kinetic updates are fast in the worst case, *efficient* if the number of events is small when compared to the number of intrinsic changes the geometric property has to go through in the worst case, *local* if each input point is associated with a small number of certificates, and *compact* if the total number of certificates and the size of the data structure is small.

Within the KDS framework, Steiner point selection becomes a hard problem; many that are suitable when input points do not move are hard to adapt effectively in the kinetic setting. One such example is the quadtree method, where the corners of the quadtree squares are inserted as Steiner points [4]. Another very commonly used such example is the circumcenters of low quality triangles [6, 8]. Aside from these very common examples, Üngör defines a more local type of Steiner points called off-centers [10], Hudson and Türkoğlu propose a local region to pick Steiner points from, a region called clipped Voronoi cells, one which includes off-centers [7], and Acar et al. choose Steiner points as geometric translations of input points at geometrically increasing distances [2]. In this paper, we use the kinetic mesh of Acar et al. as a point location structure and extend the clipped Voronoi computations of Hudson and Türkoğlu to the kinetic setting.

*Department of Computer Science, University of Chicago

Our contribution in this paper is a first step towards solving the kinetic mesh refinement problem for achieving arbitrary quality triangulations. Building on the result of Hudson and Türkoğlu [7], we design a KDS for computing clipped Voronoi cells so that a meshing algorithm can use it for picking Steiner points to construct the output mesh. Given a point v, our KDS provides the following procedures pertaining to v: ApproximateNN(v) to compute an approximation of the distance from v to its nearest neighbor, ClippedVoronoi(v, β) to compute the clipped Voronoi cell of v, and AddVertex(p, v) to insert into the output a Steiner point p near v. We prove in Theorem 3 that all of the above procedures run in constant time, create a constant number of certificates, and hence can be updated in constant time upon a certificate failure. Our KDS reduces the problem of kinetic mesh refinement of arbitrary quality to suitable choice of Steiner points within local neighborhoods.

2 Definitions

In this section we provide preliminary definitions for the rest of the paper. To distinguish mesh points from any point in space we refer to mesh points as vertices.

Definition 1 *Given a vertex v, $\mathrm{NN}(v)$ is the distance from v to its nearest neighbor, and $\mathrm{Vor}(v)$ is the Voronoi cell of v consisting of all points x such that for any vertex $u \neq v$, $|ux| \geq |vx|$. Then v is called ρ-well-spaced [9] if $\mathrm{Vor}(v)$ is inside the ball centered at v with radius $\rho\,\mathrm{NN}(v)$, and a mesh is called ρ-well-spaced if every vertex in the mesh is ρ-well-spaced.*

Using the fact that the dual of a Voronoi diagram yields a Delaunay triangulation, one can observe that well-spacedness is equivalent to large minimum angles in the output triangulation; the lower the parameter ρ, the larger the minimum angles.

Definition 2 *[7] The β-clipped Voronoi cell of a vertex v, $\mathrm{Vor}^{\beta}(v)$, is the intersection of $\mathrm{Vor}(v)$ and the ball centered at v with radius $\beta\,\mathrm{NN}(v)$. For any point $x \in \mathrm{Vor}^{\beta}(v)$, the ball centered at x with radius $|vx|$ is a witness[1] ball empty of vertices, and the witness region of $\mathrm{Vor}^{\beta}(v)$ is the union of witness balls.*

Figure 1 depicts the above definitions of well-spacedness, clipped Voronoi cells, witness balls and witness regions.

Definition 3 *[8] Given a point x in space, the local feature size of x, $\mathrm{lfs}(x)$, is the distance from x to its second-nearest input vertex, and a size-conforming mesh is one in which for every output vertex v, $\mathrm{NN}(v) \in \Theta(\mathrm{lfs}(v))$; Ruppert proves that being size-optimal is equivalent to being size-conforming.*

[1] The origial term used in [7] is "certificate", however, we use the term "witness" to avoid confusion with KDS certificates.

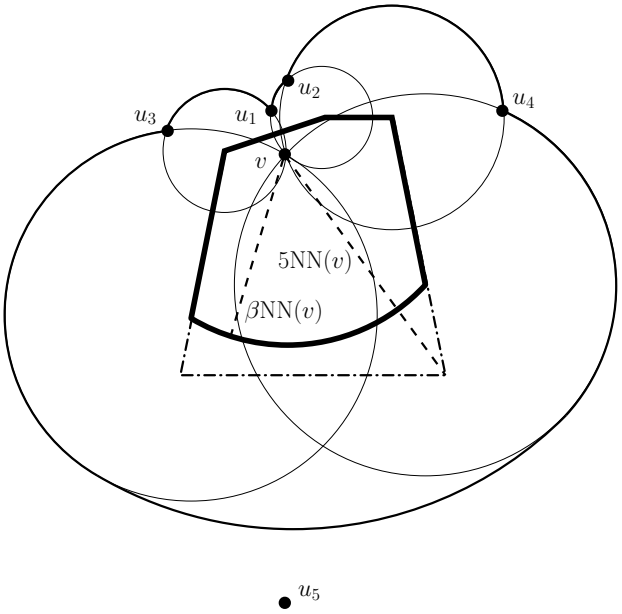


Figure 1: A vertex v and its neighbors u_1 through u_5. Farthest point in $\mathrm{Vor}(v)$ is at $5\,\mathrm{NN}(v)$ distance, thus, v is 5-well-spaced but not β-well-spaced. As a result, $\mathrm{Vor}^{\beta}(v)$ is shown with a thick boundary and its witness balls and witness region with a thin boundary.

Definition 4 *A mesh refinement algorithm is bottom-up if it incrementally ensures well-spacedness of the vertices in the order of their local feature size. For a given constant γ, a bottom-up algorithm inserts a Steiner vertex u, when every mesh vertex v with $\mathrm{NN}(v) < \gamma\,\mathrm{NN}(u)$ is ρ-well-spaced.*

Our data structure requires a bottom-up meshing algorithm that outputs a size-conforming mesh to ensure local searches and fast runtimes.

3 Data Structure and Implementation

Hudson and Türkoğlu use quadtrees for point location as they provide cruicial properties to guarantee fast runtimes. However, quadtrees do not adapt well to the kinetic setting, and we instead use the kinetic mesh of Acar et al. as our point location structure [2]. Their KDS picks Steiner points from points they call *satellites*, pre-defined geometric translations of input vertices at geometrically increasing distances. They maintain the Voronoi diagram of their output vertex set and their Voronoi cells, or *cells* in short so as to distinguish from the Voronoi cells of our definition, provide properties similar to those of the quadtree squares, and our guarantees depend on them:

(A) Each cell has a constant number of neighbors.

(B) The size of each cell is in proportion to the local feature size of the points in the cell.

The size of a cell can be described by the nearest neighbor distance of its node with respect to other nodes in the diagram, and the well-spacedness of the node guarantees that the whole cell is contained within a ball of constantly larger radius, less than 9/2 times larger to be precise. As shown in Lemma 3.6 [2], the nearest neighbor is at distance $> 2^{\ell-2}$, where ℓ, defined as the *rank* of the node, is already computed by the data structure. Using the rank information of the nodes, we implement our procedures as follows:

- ApproximateNN(v) returns the distance $2^{\ell-2}$, where ℓ is the rank of v. This procedure does not create any certificates.
- AddVertex(p, v) starts at the cell of the node that contains v and iteratively advances by moving the search to the neighbor node closest to p. Finally, it reaches the cell that contains p as none of its neighbor nodes is closer to p, and inserts p in that cell as a Steiner vertex.

We certify this procedure by the distance comparisons between the final cell and its neighbors which create a set of what we call *cell* certificates. Upon the failure of a cell certificate, we update this search by restarting it from the node of the failure.

- ClippedVoronoi(v, β) performs in two stages; in the first stage it collects information about nearby vertices. It starts at the cell of the node that contains v and explores the region by moving outward along neighboring cells, in increasing order of their nodes' distances from v. It continues the search to find the nearest neighbor of v and to determine $\mathrm{NN}(v)$. Then it continues the search to explore other vertices within $2\beta\,\mathrm{NN}(v)$ distance of v. It stops exploring further at a cell if one of the two stopping conditions is satisfied:

  1. The distance from v to the cell is too large.
  2. The size of the cell is too small.

  Once the exploration of the nearby vertices is over, it proceeds to the second stage to determine actual Voronoi neighbors among the vertices within $2\beta\,\mathrm{NN}(v)$ distance. It removes a vertex u from the set if there is no witness ball incident to v and u empty of vertices or if the radius of the smallest one is larger than $\beta\,\mathrm{NN}(v)$ (Figure 2). It returns the remaining set as $\mathrm{Vor}^{\beta}(v)$.

To make the implementation of the procedure ClippedVoronoi(v, β) more precise, we introduce two constants λ_1 and λ_2 for the stopping conditions. In the first condition, we define the distance from v to a cell with node p to be too large if $|vp| > (2\beta+\lambda_1)\,\mathrm{NN}(v)$ (Lemma 1). And in the second condition, we define the size of a cell to be too small if the rank of its node is less than $\ell - \lambda_2$, where ℓ is the rank of v (Lemma 2).

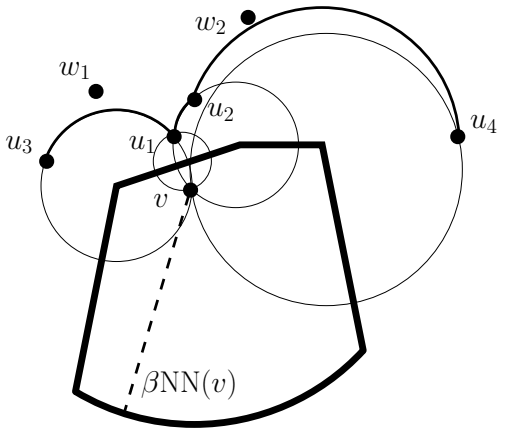


Figure 2: Setting of Figure 1. $\mathrm{Vor}^{\beta}(v)$ is represented by $\{u_1, u_2, u_3, u_4\}$. Smallest witness balls incident to each Voronoi neighbor u_i and v is depicted. For the vertices w_1 and w_2 within $2\beta\,\mathrm{NN}(v)$ distance there are no witness balls. Each of the circumcircles defined by the triplets $\{v, w_1, u_i\}$ and $\{v, w_2, u_i\}$ contains at least one other vertex.

Finally, we summarize the certificates and the kinetic update of ClippedVoronoi(v, β). In the first stage, distance comparisons create what we call *search* certificates. And, in the second stage, distance to circle comparisons create what we call *voronoi* certificates. Upon the failure of any one certificate, we restart that stage from scratch.

4 Proofs

In this section, we first prove that there exists constants λ_1 and λ_2 for our procedures; we then prove that our KDS is effective. Our proofs rely on the assumptions that the mesh refinement algorithm that employs our KDS is bottom-up and that its output is size-conforming.

The underlying summary of the proofs in this section is that local feature size is a smooth function and working with size-conforming structures also achieve similar smoothness properties. The first Lemma is based on the property that lfs cannot get too large within a locality.

Lemma 1 *There exists a constant λ_1 such that if the procedure* ClippedVoronoi(v, β) *stops exploring a cell because of the first condition, the distance from v to any point in that cell is greater than $2\beta\,\mathrm{NN}(v)$.*

Proof. For any point x within $2\beta\,\mathrm{NN}(v)$ distance of v, using the triangle inequality, we know that there

exist two output vertices within $O(\mathrm{NN}(v))$ distance of x, namely v and the nearest neighbor of v. And, since we assume that the output is size-conforming, we have $\mathrm{lfs}(x) \in O(\mathrm{NN}(v))$. Then condition (B) implies that the cell that includes x has size bounded by $O(\mathrm{NN}(v))$. Let λ_1 be the constant hidden in the big-Oh notation and let p be the node of that cell. Using the triangle inequality we have $|vp| \leq |vx| + |xp|$ or $|vp| \leq (2\beta + \lambda_1)\,\mathrm{NN}(v)$. Proof follows from the contrapositive argument. □

The below Lemma is based on the property that lfs stays large within large empty balls. In other words, if lfs is too small in some local neighborhood, then there must be vertices in between to support smoothness, and well-spacedness is sufficient to ensure that.

Lemma 2 *Assuming that for every mesh vertex u with $\mathrm{NN}(u) < \gamma\,\mathrm{NN}(v)$ is ρ-well-spaced, there exists a constant λ_2 such that the union of the cells explored by the procedure* ClippedVoronoi(v, β) *covers the witness region of* $\mathrm{Vor}^{\beta}(v)$.

Proof. Given a cell at which ClippedVoronoi(v, β) stops exploration, it is sufficient to prove that the cell does not intersect the witness region of $\mathrm{Vor}^{\beta}(v)$. If the reason to stop is the first condition, by Lemma 1, we know that the cell lies at a distance more than $2\beta\,\mathrm{NN}(v)$ away from v, thus it cannot intersect the witness region. If the reason to stop is the second condition, for any point p in the witness region, we know that p lies in a ball of radius at least $\mathrm{NN}(v)/2$. Therefore the premise of this Lemma satisfies the premise of Theorem 3 in [7]. As a result, we have $\mathrm{lfs}(p) \in \Omega(\mathrm{NN}(v))$ or $\mathrm{lfs}(p) \in \Omega(2^{\ell})$. This means that there exists a constant λ_2 large enough such that by condition (B), for any cell of rank less than $\ell - \lambda_2$, the local feature size of any point in the cell is less than $\mathrm{lfs}(p)$, which is to say those cells cannot intersect the witness region. □

Since lfs is a smooth function and since we ensure that the search does not extend to neighborhoods that have small local feature sizes, packing arguments prove that our procedures take constant time.

Theorem 3 *In our KDS,* ClippedVoronoi(v, β) *maintains* $\mathrm{Vor}^{\beta}(v)$ *and* AddVertex(p, v) *maintains insertion of a Steiner vertex p in constant time.*

Proof. The proof for ClippedVoronoi(v, β) relies on two facts: the search explores only a constant number of cells and each cell contains a constant number of vertices. For the first fact we use a packing argument: Lemma 1 bounds λ_1 which in turn proves that the search does not go beyond a distance of $O(\mathrm{NN}(v))$, and on the other hand, Lemma 2 bounds λ_2 which in turn proves that each cell explored is of size at least $\Omega(\mathrm{NN}(v))$. For the second fact we use another packing argument: for a given cell, condition (B) states that the size of the cell is bounded by $O(\mathrm{lfs}(p))$ for any given point p inside the cell, and since the mesh is size-conforming, any vertex p inside the cell has an empty ball of size $\Omega(\mathrm{lfs}(p))$.

The proof for AddVertex(p, v) is simpler: p must lie within the cells explored in the ClippedVoronoi(v, β) call, thus the search for p involves only a subset of those cells. □

5 Conclusion

In this paper, we showed how to support a class of mesh refinement algorithms with a kinetic data structure that could help find appropriate Steiner points in constant time. For simplicity, we have chosen to explain the algorithms in a purely theoretical fashion, however, we can employ several improvements that will make the constants in our algorithms practical.

References

[1] U. A. Acar, A. Cotter, B. Hudson, and D. Türkoğlu. Dynamic well-spaced point sets. In *SCG '10: Proceedings of the 26th Annual Symposium on Computational Geometry*, pages 314–323, 2010.

[2] U. A. Acar, B. Hudson, and D. Türkoğlu. Kinetic mesh refinement in 2d. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11, pages 341–350, 2011.

[3] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[4] M. Bern, D. Eppstein, and J. R. Gilbert. Provably Good Mesh Generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[5] S. Har-Peled and A. Üngör. A time-optimal Delaunay refinement algorithm in two dimensions. In *21st Symposium on Computational Geometry*, pages 228–236, 2005.

[6] B. Hudson, G. L. Miller, and T. Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356, 2006. Long version in Carnegie Mellon University Tech. Report CMU-CS-06-132.

[7] B. Hudson and D. Türkoğlu. An efficient query structure for mesh refinement. In *Canadian Conference on Computational Geometry*, pages 115–118, 2008.

[8] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[9] D. Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, Pittsburgh, August 1997.

[10] A. Üngör. Off-centers: A new type of Steiner point for computing size-optimal quality-guaranteed Delaunay triangulations. In *LATIN*, pages 152–161, 2004.

Dynamic Convex Hull for Simple Polygonal Chains in Constant Amortized Time per Update

Norbert Bus* Lilian Buzer*

Abstract

We present a new algorithm to construct a dynamic convex hull in the Euclidean plane, supporting insertion and deletion of points. Both operations require amortized constant time. At each step the vertices of the convex hull are accessible in constant time. The algorithm is on-line, does not require prior knowledge of all the points. The only assumptions are that the points have to be located on a simple polygonal chain and that the insertions and deletions must be carried out in the order induced by the polygonal chain.

1 Introduction

The convex hull of a set of n points in a Euclidean space is the smallest convex set containing all the points. Constructing the convex hull of a point set is a fundamental problem in computational geometry. It has applications in, e.g., pattern recognition, image processing and micro-magnetic analysis [2, 6]. Many problems can be reduced to determining the convex hull of a point set, such as Delaunay triangulation and half space intersection [5]. Therefore developing robust and efficient algorithms for the core problem has received much attention. If one considers the real-RAM model, an optimal output sensitive algorithm to construct the convex hull of n points in a plane was published in [7] having $O(n \log h)$ time complexity where h is the output size. If the point set is a simple polygonal chain, the best algorithm, a result of Melkman, runs in linear time [1]. If one requires the data structure to be dynamic, namely to handle insertions and deletions of arbitrary points an optimal algorithm requiring $O(\log n)$ time for both operations was proposed in [3]. Changing the computational model to the word-RAM model and using Graham's scan [8] to construct a convex hull the running time is essentially the time to sort the points, taking, e.g., $O(n \log \log n)$ time [9]. Dynamic data structures supporting deletion and insertion in the word-RAM model require an optimal $O(\frac{\log n}{\log \log n})$ time for both operations assuming that word length is $\Theta(\log n)$, see [4].

*Université Paris-Est, LIGM, A3SI-ESIEE, France, {busn, buzerl}@esiee.fr

Our contribution In this paper we give an on-line algorithm to construct the dynamic convex hull of a simple polygonal chain in the Euclidean plane supporting deletion of points from the back of the chain and insertion of points in the front of the chain. Both operations require amortized constant time considering the real-RAM model. The main idea of the algorithm is to maintain two convex hulls, one for efficiently handling insertions and one for deletions. These two hulls constitute the convex hull of the polygonal chain.

2 Overview of our algorithm

Our algorithm works in phases. For a precise formulation let us first define some necessary notations. A polygonal chain S in the Euclidean plane, with n vertices, is defined as an ordered list of vertices $S = (p_1, p_2, \ldots, p_n)$ such that any two consecutive vertices, p_i and p_{i+1} are connected by a line segment. A polygonal chain is called simple when it is not self-intersecting. For simplicity, we assume that the points are in general position. Our algorithm handles insertion and deletion of points into the current convex hull in the order induced by S. This results in the fact that the current convex hull always contains a contiguous subchain of S, let us denote it by $S_i^j = (p_i, \cdots, p_j)$ and the points are effectively inserted/deleted in a FIFO manner. Let us denote the convex hull of S_i^j with C_i^j. Therefore, given a convex hull C_i^j, inserting a point results in C_i^{j+1} while removing the first point results in C_{i+1}^j.

At the beginning of each phase, we initialize a simple data structure called the *phase convex hull* that maintains the representation of the convex hull of a subrange of the polygonal chain. Each phase handles an arbitrary number of insertions and handles deleting the points that were present when the phase started. Assuming that the phase convex hull first covered S_a^b this means we can delete the points $p_a \ldots p_b$. A phase ends, when we first delete a point that was not covered by the initial convex hull. After that, a new phase starts and we initialize a new phase convex hull. See Figure 1.

We state the main result of our algorithm in Theorem 1.

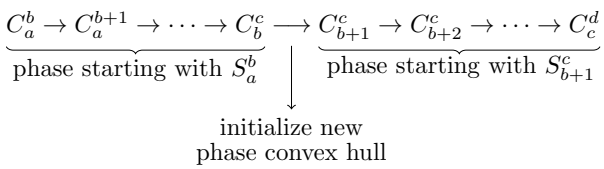


Figure 1: Example of two phases

Theorem 1 *The amortized time complexity of insertion and deletion of points in a convex hull of a simple polygonal chain is constant.*

Proof. Assume that each point has been inserted and later removed. In Section 6 we show that the $i-th$ phase runs in $O(k_i + l_i)$ time where k_i is the number of insertions in the phase and l_i is the number of deletions. During the whole algorithm each point has been inserted and deleted exactly once hence there have been n insertions and n deletions overall. Therefore the overall running time of the phases is $O(n)$, yielding the desired result. □

3 Convex hulls

In this section we introduce the phase convex hull representing the convex hull of the polygonal chain. Furthermore, we describe two convex hulls that are maintained during the phase. These two hulls' purpose is to enable that insertion and deletion of points for the phase convex hull run in constant amortized time. We suggest that the reader is familiar with the Melkman algorithm [1] as our method builds heavily on it. Briefly, the method builds the convex hull of a polygonal chain by iteratively (in the proper order) adding the points to the convex hull and modifying it as necessary. At the beginning of the phase let S_a^b be the current polygonal chain while at the end let it be S_b^c. Let us denote by S_i^j the polygonal chain at an arbitrary step during the phase and C_i^j the corresponding convex hull.

Phase convex hull The *phase convex hull* denoted by C^* is the data structure representing the convex hull C_i^j, containing all of its points in two dequeues. Every point is contained in exactly one of the two dequeues except for two: the front of both dequeues refer to the same point of the subchain, the one contained in C_i^j with highest index and similarly, the back of both dequeues refer to the same point of the subchain, the one contained in C_i^j with lowest index. We refer to the front of both dequeues as front opening and to the back as back opening. Connecting the two dequeues gives the ordered circular list of points in C_i^j. See Figure 2. Clearly, if one considers the back opening to be *closed* (i.e., as if being glued together, removing the duplicate copy of the back point), one has the data structure used in the Melkman algorithm. At the beginning of the phase C^* is built for S_a^b using the Melkman algorithm.

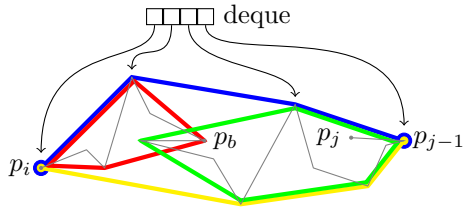


Figure 2: The two dequeues (blue and yellow) constituting C^*. For clarity we include one schematically. In green and red we depict C^+ and C^- respectively.

The following data structures aid to handle insertion or more importantly deletion of points (since the phase convex hull itself could handle insertions).

Incremental convex hull The *incremental convex hull* is a convex hull incrementally built with the Melkman algorithm for the points in the polygonal chain added after the initialization of the phase. Let us denote it by C^+. At any step S_i^j, formally, C^+ is the same as C_{b+1}^j.

Decremental convex hull The *decremental convex hull* is a convex hull built with the Melkman algorithm for the points present at the beginning of the phase but according to the reverse order of the points. Let us denote it by C^-. At each deletion a point is removed from C^-. At any step S_i^j, formally, C^- is the convex hull of the points S_b^i (note the reverse order). This data structure has to maintain additional information that will be used for efficient deletion of points. First, while creating C^-, for each point in S_b^a, the list of points that were removed from the previous convex hull should be kept. Let us call these points the *history* of a point. This enables that the algorithm can be 'rewound', i.e, the points of C^- can be deleted in a FILO order. As C^- was built in the reverse order this is exactly the deletion order we need. Second, the polygonal regions defined by $C_k^j \backslash C_{k+1}^j$ for $a \leq k \leq b$, in other words the difference between consecutive convex hulls in the Melkman algorithm for building C^-, have to be kept. At the beginnning of each phase we build the decremental convex hull for S_b^a. In Figure 3 we show the regions where the red polygonal chain is S_a^b. The green line corresponds to the polygonal chain of the points inserted during the phase.

We will need a simple property of the regions, namely that they form an ordered partitioning of the plane that enables a certain operation. See Lemma 2.

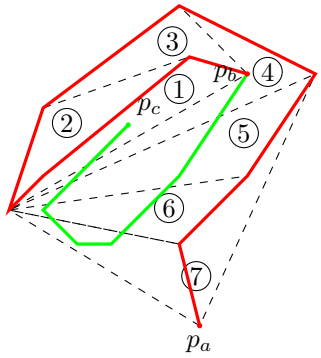


Figure 3: Regions.

Lemma 2 *While constructing C^- for S_a^b one can create an ordered list of regions in $O(b-a)$ time. Such a partitioning enables the maintenance of the highest ordered region that contains any point inserted during the phase. This operation takes $O(c-b)$ time for a phase.*

In Figure 3 the highest ordered region containing a point inserted in the phase is region no.7. The lemma is a straightforward consequence of the Melkman algorithm as given the current highest ordered region containing an inserted point one only has to check whether a newly inserted point is contained in the region following (in order) the current one. One has to be careful when adding the point p_{b+1} as there was no highest ordered region before, but this case is trivial.

4 Insertion

In this section we describe the method to handle the insertion of p_{j+1} into the convex hull of S_i^j.

In order to insert the point p_{j+1} into C^* it is sufficient to do one step of the Melkman algorithm at the front opening of the dequeues. For that, consider the back opening of C^* to be closed, i.e., the two dequeues behave like one. One has to be cautious when the Melkman algorithm deletes the point being the back opening as the new back opening should be by our definition the point in C_i^j with least index. C^+ has to be updated with p_{j+1} as well, which is done using the Melkman algorithm. Moreover, as long as the front opening is one of the points in C^- one has to update the highest ordered region of C^- containing any inserted point.

5 Deletion

In this section we describe the deletion of the point p_i from S_i^j. We will need a property concerning the points in the phase convex hull C^*. Note, that the points of C^* either belong to C^+ or C^-. The following lemma describes how the points' distribution (with respect to whether they belong to C^+ or C^-) changes during the execution of the operations in a phase. It states that there cannot be arbitrary distributions, e.g., points from C^+ and C^- in an alternating order.

Lemma 3 *The points in C^* are partitioned into contiguous ranges according to whether they belonging to C^- or C^+. At any step in a phase there are at most two such partitions, one containing points of C^+ and one containing points of C^-. The partitioning changes in a specific pattern during a phase: at the beginning there are only points from C^- in C^*; then two partitions; finally only points from C^+ are located in C^*.*

The fact that C^* is partitioned into at most two parts is a consequence of the simplicity of the polygonal chain. The strict order also follows easily since C^+ is monotonically growing while C^- is monotonically shrinking.

To delete the point p_i there are several scenarios that have to be handled differently. As a common point in all cases p_i has to be removed from C^- and its history has to be added to C^- ('rewinding' the Melkman algorithm). Let us first group the different cases according to Lemma 3.

Case 1: If the phase convex hull contains only points from C^- then one can simply remove p_i from C^* and add the points of the history of p_i to C^*. This is valid as long as there shall be no point of C^+ added to C^*. The highest ordered region is maintained exactly for checking this. As long as the deleted region has no points assigned to it we can proceed as explained. If it is not empty then an expensive operation is required, namely to add all the points of C^+ to C^*. This can be done with the Melkman algorithm considering the back opening to be closed. Even though this operation might require linear time in the number of insertions it can happen only once for each phase therefore its amortized time complexity is constant.

Case 2: If the phase convex hull contains points from both C^+ and C^-, we have the most complicated case. Obviously the point p_i to be removed is in C^-. Let us denote the neighbors of p_i in C^* by x and y. The edge between x and y would become an edge of C^* if there are no points in the triangle defined by p_i, x, y. But usually this is not the case therefore one has to create new edges of C^* that correspond to the vertices located in this triangle. Adding these points (and at the same time checking if there are points inside the triangle) is done as follows. We further categorize this case into three sub cases depending on the neighbors of p_i.

Case 2a: If both x and y belong to C^- then clearly C^* can only be modified by the points from the history of the currently deleted point p_i. In such a situation using the Melkman algorithm one can add the ordered history of p_i to C^*.

Case 2b: If one point, e.g., x belongs to C^+ then there might be points of C^+ that have to be inserted into C^*. In this case first the history of p_i and its neighboring point in C^- (that is not y) should be inserted into C^* and then starting from x we shall add the vertices of C^+ in the circular order (starting with the neighbor of x not contained in C^*) using the Melkman algorithm but just as long as they create new vertices on C^*. One can show that if a point of C^+ is inside C^* then there are no other points that shall be inserted.

Case 2c: If both x and y belong to C^+ then a similar process has to be carried out namely first inserting the points of the history (with the 2 neighbors of p_i in C^-) and then the vertices of C^+ in the proper circular order starting from x and y. Note that x and y define two different parts of C^+ that have to be inserted into C^* and to maintain a low running time one has to insert points from these two parts in an alternating order (one cannot proceed with points from the part of x after finishing the points starting from y). When a point remains inside the phase convex hull we can finish inserting points from its part. See Figure 4 for a schematic illustration of the process. In order to be able to utilize the Melkman algorithm the added points have to belong to a simple polygonal chain, otherwise using Melkman would be impossible.

Case 3: If C^* contains only points of C^+ than $C^- \subset C^+$ therefore there is no change in C^*.

How to ensure the applicability of the Melkman algorithm and the proof of correctness, e.g, why is it sufficient to add only the history of p_i in Case 2 is left for the full paper.

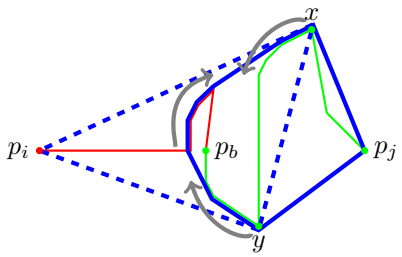


Figure 4: Deleting the point p_i. Solid blue lines denote the convex hull after deleting p_i. Gray arrows denote the points that have to be inserted into C^*.

6 Complexity

Let us now show that each phase takes time linear in the number of insertions and deletions. Let us denote them by k and l respectively.

Initialization: Clearly, initializing C^*, C^+ and C^- is linear in the number of points present at the beginning of the phase since we only utilize a modified Melkman algorithm. This indeed is the same as the number of points deleted in the phase. Therefore the complexity of initialization is $O(l)$.

Insertion: Using the Melkman algorithm for both the phase convex hull and the incremental convex hull takes $O(k)$ time. During insertion of points one has to also maintain the highest ordered region containing any point of C^+. This can be done in $O(k)$ time.

Deletion: Clearly, during executing the deletions, the number of points inserted into C^* using the Melkman algorithm is not more than a constant times the points in C^- which is $O(l)$ and the points of C^+ added to C^* which is less than $O(k)$ as no point can reappear on C^* due to the monotonicity of the operations.

This results in the following theorem.

Theorem 4 *The running time of one phase is $O(k+l)$ given that there are k insertions and l deletions.*

7 Conclusion

We have presented an algorithm that handles insertion and deletion of points from/into the convex hull of a simple polygonal chain. Each operation takes amortized constant time. The operations are carried out in a FIFO manner, namely that points are inserted on one end of the chain while points are deleted from the other. It would be interesting to see if this constraint can be removed, namely that points can be inserted/deleted on both ends of the chain.

References

[1] A. A. Melkman. *On-line construction of the convex hull of a simple polyline* Information Processing Letters, 1987.

[2] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction* Springer-Verlag, 1985.

[3] G. S. Brodal and R. Jacob. *Dynamic Planar Convex Hull* FOCS, 2002.

[4] E. D. Demaine and M. Patrascu. *Tight Bounds for Dynamic Convex Hull Queries (Again)* SOCG, 2007.

[5] F. Aurenhammer. *Voronoi Diagrams – a Survey of a Fundamental Geometric Data Structure* ACM Computing Surveys, 1991.

[6] D. G. Porter, E. Glavinas, P. Dhagat, J. A. O'Sullivan, R. S. Indeck and M. W. Muller. *Irregular grain structure in micromagnetic simulation* Journal of Applied Physics, 1996.

[7] T. M. Chan. *Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions* Discrete and Computational Geometry, 1996.

[8] R. L. Graham. *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set* Information Processing Letters, 1972.

[9] Y. Han. *Deterministic sorting in $O(n \log\log n)$ time and linear space* Journal of Algorithms, 2004.

A local geometry based algorithm for point cloud edge classification *

Mihai Sorin Stupariu†

Abstract

The identification of salient features in point clouds is one of the main application areas of geometric algorithms. In this paper, we deal with identifying edges in high density point clouds on polyhedral objects on the basis of a two-stage classification method. The first step consists in applying Principal Component Analysis, which commonly uses the eigenvalues of the covariance matrix and provides several point classes. In the second stage, a neighbourhood-based analysis is used for refining the classification. The method relies on the discrete Gaussian curvature, defined as a weighted angular defect and computed for the vertices of a suitably chosen triangulation. A special emphasis is put on identifying jump edges, which correspond to surface discontinuities and crease edges, which occur where two planes of the same object meet. The key issue is that the local geometry of a point cloud is different in a neighbourhood of a jump edge - planar structure, compared to a crease edge, where two planes intersect along the support line of the edge.

1 Introduction

Even though formally unorganized, a point cloud carries in many situations an internal structure and fundamental shapes can be detected (e.g. Schnabel [11]; van Lankveld et al. [14]). The hidden geometric structure can be unveiled by using appropriate tools, such as the ones provided by multivariate statistics. As shown by Hoppe et al. [6], Principal Component Analysis (PCA for shorthand) applied to a point cloud yields a well-defined local structure in the neighbourhood of each point. The technique may be completed by other methods, in order to extract salient features or to achieve a suitable classification. Thus, Gumhold et al. [4] and Pauly et al. [8] computed a minimum spanning graph for approximating the features of interest. Roggero [10] derived moments and tensor of inertia from the outcomes of the eigenvalue analysis, in a region-growing approach. Cao et al. [1] performed a tensor voting technique to refine normal estimates, which were derived by using PCA. The random sample consensus algorithm was used by Yang et al. [16] for segmenting building objects identified through the multivariate statistical approach. Other complementary techniques include gridded approximate nearest neighbour searches for fast classification of geometric features (Shi and Zakhor [12]) or 3D shape analysis and region growing (Carlberg et al. [2]).

Another way to tackle classification problems is to define an additional structure in the point cloud and to apply algorithms suited to structured data (e.g. Remondino [9]). For instance, a triangulation provides a geometric structure to an unorganized point cloud. Consequently, in this framework, one can derive geometric measures useful in characterizing important features of the cloud. Gorte [3] adapted an iterative, raster-based algorithm, controlled by a single parameter related to dissimilarity of adjacent segments. A clustering technique was used by Hofmann [5] for analysing a 3D triangle mesh parameter space. Progressive TIN densification stands as one of the classic methods for filtering point cloud data, as pointed by Zhang and Lin [17]. Tse et al. [13] proposed a method based on the Delaunay triangulation and its dual Voronoi diagram aiming to locate structures such as building blocks and roofs.

The goal of this paper is to present a classification method combining the two approaches and aiming to identify edges in high density point clouds. Specifically, by using the outcomes of the Principal Component Analysis, we construct a local triangle mesh. Then, we show that the classification provided by the multivariate statistics can be refined by using appropriate geometric quantities derived from the triangulation.

2 The algorithm

The algorithm has two stages, both based on neighbourhood analyses (Figure 1). In brief, the first step relies on eigenanalysis, while the second one is based on geometric information.

A particular goal of the classification is to make it possible to distinguish between points belonging to jump edges and those situated on crease edges. Both edge types constitute salient features of a point cloud, but they are different in nature: the former class corresponds to height or depth discontinuities of the sur-

*This work was supported by the Swiss Enlargement Contribution in the framework of the Romanian-Swiss Research Programme, project WindLand, project code: IZERZO 142168/1 and 22 RO-CH/RSRP.

†University of Bucharest, Faculty of Mathematics and Computer Science and Institute of Research of University of Bucharest ICUB; Transdisciplinary Research Centre Landscape-Territory-Information Systems, CeLTIS

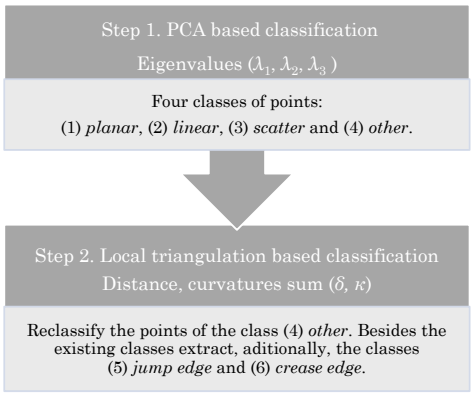


Figure 1: The two stages of the algorithm.

face, while the latter occurs where two planes, which may belong to the same object or to adjacent objects, meet (e.g., Wang and Shan [15]). The main hypothesis tested in the study is that the local geometry of a point cloud differs for jump edges compared to crease edge. Thus, in the neighbourhood of a jump edge, the local structure is close to a planar one, whereas for crease edges two planes intersecting along the support line of the edge.

In the sequel let $\mathcal{C}$ denote a point cloud situated in the space $\mathbb{R}^3$.

First step - PCA based classification

One can make a substantial step towards understanding the structure of the cloud $\mathcal{C}$ around a fixed point P by investigating the spatial distribution statistics. Thus, the Principal Component Analysis, based on analysing the eigenvalues of the covariance matrix, identifies the main directions of variation of the data. For instance, if around P the points are situated close to a plane, the PCA will provide two principal components that account together for a significant proportion of the variation, that is the first two eigenvalues are comparable in size and much bigger than the third one. This approach is implemented as follows. Consider a fixed a search radius $r > 0$ and extract the sub-cloud $\mathcal{S}$ of neighbours of P inside the ball of centre P and radius r. Let $\lambda_1, \lambda_2, \lambda_3$ be eigenvalues of the covariance matrix ($\lambda_1 \geq \lambda_2 \geq \lambda_3$). Comparisons of the eigenvalues yield the following point classes (cf. [2], [12])):

1. *planar point* if $\lambda_1 \approx \lambda_2 >> \lambda_3$;
2. *linear point* if $\lambda_1 >> \lambda_2 \approx \lambda_3$;
3. *scatter point* if $\lambda_1 \approx \lambda_2 \approx \lambda_3$;
4. *other*, otherwise.

From a practical perspective, the classification above can be implemented by using appropriate thresholds (chosen by the user): t_1 for a ratio 'close' to 1, t_2 for 'small' and t_3 for 'large'. With these notations, the classification can be phrased as follows:

1'. *planar point* if $\lambda_2/\lambda_1 > t_1$ and $\lambda_3 < t_2$;

2'. *linear point* $\lambda_3/\lambda_2 > t_1$ and $\lambda_1 > t_3$;

3'. *scatter point* $\lambda_2/\lambda_1 > t_1$ and $\lambda_3/\lambda_2 > t_1$.

Second step - local triangulation

The aim of the second step is to refine the classification provided by the PCA-based one and relies on geometric tools. For achieving this aim, a topological structure has to be introduced in the point cloud. A possible approach, further explored in this paper, is based on the use of triangulations. Instead of generating a global triangulation, we exploit the information provided by the multivariate statistics and generate local triangle meshes, using the plane determined by the first two principal axes as reference plane. Specifically, let P be a point classified as *other*. Consider again the sub-cloud $\mathcal{S}$ inside the ball of centre P and radius r. We generate a 2.5 Delaunay triangulation $\mathcal{T}$ for the points of the sub-cloud $\mathcal{S}$ by considering P as origin and the plane $z = 0$ as being the plane $\pi_P^{1,2}$ determined by the first two principal axes. This means that we project the points of $\mathcal{S}$ on the plane, we generate the 2D Delaunay triangulation for the set $\mathcal{S}'$ of projected points and we then lift the topological structure back to the cloud $\mathcal{S}$.

For the triangle mesh $\mathcal{T}$ we compute the following geometric quantities:

- δ = the distance from P to the border of the convex hull of the planar point set $\mathcal{S}'$,
- κ = the sum of the Gaussian curvatures of the vertices of $\mathcal{T}$ that are not lying on the border (the Gaussian curvature at a vertex is defined as a weighted angular defect, e.g. [7]).

The choice of these parameters is related to the following hypotheses: (i) edge points are closer to the border of the convex hull than other types of points; (ii) for an 'interior' point or for a point in the neighbourhood of a jump edge, the triangle mesh is almost planar and hence the deviation from planarity (with respect to the plane $z = 0$) of $\mathcal{S}$ is very small, that is the sum of Gaussian curvatures κ is close to 0. The second step of the algorithm yields the following point classes, defined by using suitable thresholds. Let $t_4 > 0$ be the threshold for 'small' distance and let $t_5 > 0$ be the threshold for 'small' curvature. Denote by N the number of points in $\mathcal{S}$. The point P is:

5. *jump edge point* if $\delta \leq t_4$ and $\kappa \leq t_5 N$;

6. *crease edge point* if $\delta \leq t_4$ and $\kappa > t_5 N$.

Furthermore, if $\delta > t_4$ and $\kappa \leq t_5 N$ the point belongs to the class *interior*, defined in the first step.

The definition above reads as follows. For an interior point one has a small deviation from planarity and the point is not close to the border of the locally generated 2D convex hull. In contrast, for points close to this border one may assume that they are situated on edges (or close to them). The difference between jump and crease edge points is actually induced by the local deviation from planarity, which is quantified by comparing the total sum of Gaussian curvatures κ with the product $t_5 N$.

Let us finally notice that, for points classified as *other* after the second step, one can repeat the procedure, but choosing as reference the planes $\pi_P^{2,3}$ and $\pi_P^{3,1}$ generated by the other pairs of principal axes.

3 Results

We tested the algorithm for a synthetic data set consisting of points selected through jittered sampling on the visible faces of two adjacent cuboids (Figure 2). No point was chosen on the remaining faces of the cuboids. Therefore, one obtained two possible local configurations around the edge points of the cuboids. Due to *a priori* knowledge of the point coordinates, it was possible to label points belonging and close to crease and jump edges. Specifically, we labelled as *edge points* the points inside a tubular neighbourhood of radius ρ. It is worth to notice that, in case of adjacent cuboids, one might need to split original edges into pieces of crease and jump edges (Figure 2).

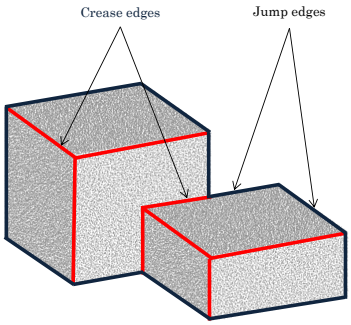


Figure 2: The synthetic data set used in the analyses. Crease and jump edges are differently represented in the figure.

The implementation of the algorithm described in section 2 is based on choosing suitable thresholds $t_1, t_2, \ldots, t_5$. Throughout the paper, the search radius r is fixed; its influence on the accuracy will be addressed in a subsequent study. Since we were mainly interested in testing hypotheses related to the local geometric behaviour of the point cloud, we estimated how the accuracy of the algorithm depends on the variation of t_4 and t_5. For the other three thresholds, we fixed the values $t_1 = 0.7, t_2 = 0.1, t_3 = 3$, on the basis of numerical experiments. With this choice, almost 91% of the interior points were correctly classified by the first step of the algorithm. In order to identify edge points, we applied the second step of the algorithm, for different values of t_4 and t_5. First, we studied the dependence between the value of t_4 and the proportion of edge points that were correctly detected by the algorithm (Figure 3). We selected four values for t_5, corresponding to different scales of flatness ($t_5 \in \{0.001, 0.01, 0.1, 1\}$). The resulting curves have a similar shape. The first inflection point, having abscissa approximately 1, is related to the density of the point cloud, while the second one, having abscissa approximately 4, is related to the choice of the radius defining the tubular neighbourhoods of the edges. The graph shows that, for $t_5 = 0.01$, one correctly identifies more than 80% of the edge points.

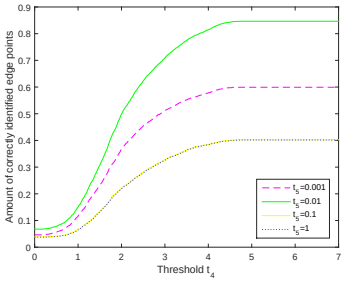


Figure 3: Relationship between t_4 and the share of correctly identified edge points. For the threshold t_5 four values were taken into account: 0.001, 0.01, 0.1, and 1, respectively.

The choice of the threshold t_5 is crucial for classifying the edge points as belonging to jump or to crease edges. If the value of t_5 is too small, almost all the edge points are classified as crease, whereas for bigger values of t_5, the number of jump edge points is overestimated (Figure 4). Nevertheless, for a suitable choice of t_5, most of the edge points are correctly classified. In particular one can distinguish between jump and crease edges having the same support line.

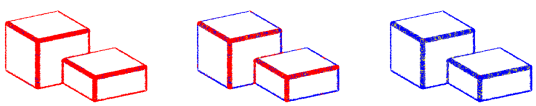


Figure 4: Choice of the threshold t_5. The threshold t_4 is fixed ($t_4 = 4.5$) and the threshold t_5 has the values 0.0001, 0.0102, and 0.12, respectively. The codes of the colours are blue=jump edge; red=crease edge.

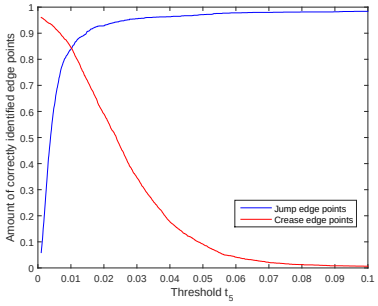


Figure 5: Relationship between t_5 and the proportion of points that are classified as belonging to jump edges or crease edges. The value of t_4 is equal to 4.5.

The heuristic approach illustrated in Figure 4 suggests that, when the value of t_5 increases, the amount of points classified as 'jump' also increases, while the opposite happens for crease edge points. It is a natural question whether the total amount of correctly identified edge points corresponding to the equilibrium is high or it is low. The specific behaviour of this variation, corresponding to a fixed value of the threshold t_4 (namely $t_4 = 4.5$) is illustrated in Figure 5. For $t_4 \approx 0.01$ approximately 85% of both jump and crease edge points are correctly identified by the algorithm. Let us finally note that actually, in the numerical tests, values of t_4 and t_5 close to 4.5, respectively to 0.01 yielded the most accurate classification of the edge points.

4 Conclusion

The study presented a classification algorithm for dense point clouds, with an emphasis on detecting jump and crease edges. The approach relied on applying Principal Component Analysis and constructing a local Delaunay triangulation, naturally associated to the point cloud and reflecting its intrinsic local geometric structure. From a quantitative perspective, besides the eigenvalues provided by the multivariate statistics, we used the local triangulation for computing the sum of discrete Gaussian curvatures and the distance to the border of a suitable 2D convex hull. The method was tested in the case of a synthetic polyhedral object, for which more than 80% of the points were correctly classified. As tasks for future work, we mention tests on real data sets and comparisons to state of the art point cloud edge detection methods.

References

[1] J. Cao, S. Wushour, X. Yao, N. Li, J. Liang, and X. Liang. Sharp feature extraction in point clouds. *IET Image Processing*, 6(7):863–869, 2012.

[2] M. Carlberg, P. Gao, G. Chen, and A. Zakhor. Classifying urban landscape in aerial LiDAR using 3D shape analysis. In *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP)*, pages 1701–1704, 2009.

[3] B. Gorte. Segmentation of TIN-structured surface models. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(4):465–469, 2002.

[4] S. Gumhold, X. Wang, and R. Macleod. Feature extraction from point clouds. In *Proc. of the 10 th International Meshing Roundtable*, pages 293–305, 2001.

[5] A. D. Hofmann. Analysis of TIN-structure parameter spaces in airborne laser scanner data for 3-d building model generation. In *IAPRS International Archives of Photogrammetry and Remote Sensing*, volume 34, pages 302–307, 2004.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. Mcdonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH 92 Proceedings)*, pages 71–78, 1992.

[7] E. Magid, O. Soldea, and E. Rivlin. A comparison of Gaussian and mean curvature estimation methods on triangular meshes of range image data. *Computer Vision and Image Understanding*, 107:139–159, 2007.

[8] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces. *Computer Graphics Forum*, 22(3):281–289, 2003.

[9] F. Remondino. From point cloud to surface: the modeling and visualization problem. *ISPRS Archives*, XXXIV-5/W10, 2003.

[10] M. Roggero. Object segmentation with region growing and principal component analysis. In *Buildings 1995, Building Department, Hong Kong*, pages 289–294, 2002.

[11] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, 2007.

[12] X. Shi and A. Zakhor. Fast approximation for geometric classification of LiDAR returns. In *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP)*, pages 2925–2928, 2011.

[13] R. Tse, C. Gold, and D. Kidner. Using the Delaunay Triangulation/ Voronoi Diagram to extract building information from raw LiDAR data. In *Proc. of the 4th International Symposium on Voronoi Diagrams in Science and Engineering*, pages 222–229, 2007.

[14] T. Van Lankveld, M. Van Kreveld, and R. Veltkamp. Identifying rectangles in laser range data for urban scene reconstruction. *Computers & Graphics*, 35(3):719–725, 2011.

[15] J. Wang and J. Shan. Segmentation of LiDAR point clouds for building extraction. In *Proceedings of the ASPRS Annual Conference*, pages 9–13, 2009.

[16] B. Yang, Z. Wei, Q. Li, and J. Li. Semiautomated building facade footprint extraction from mobile LiDAR point clouds. *IEEE Geoscience and Remote Sensing Letters*, 10(4):766–770, 2013.

[17] J. Zhang and X. Lin. Filtering airborne LiDAR data by embedding smoothness-constrained segmentation in progressive TIN densification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 81:44–59, 2013.

Improved single tree-crown extraction from 3D point clouds*

Domen Mongus† Borut Žalik

Abstract

This paper proposes a new approach for the delineation of single tree-crowns in LiDAR data that is based on Locally Fitted Surfaces (LoFS). This new definition of watershed markers is, in comparison to the traditionally used local maxima of the smoothed canopy height model, better adapted for dealing with anisotropic shapes regarding tree-crowns of various sizes. As confirmed by the results, approximately 6% increase in overall accuracy is achieved in this way.

1 Introduction

By allowing for fast and accurate monitoring of vast geographical areas with resolutions reaching beyond square decimetres, airborne Light Detection and Ranging (LiDAR) technology has triggered a new wave of research in Earth observations. The unprecedented information contained within the acquired datasets, however, impose great challenges for data processing and drive the demand for more efficient object recognition algorithms capable of dealing with those topologically unstructured and often noisy data. With the obviously huge environmental impacts of forests, capturing and processing forest data is an especially important yet difficult task due to their vast extents and the complex geometries of trees.

By identifying and delineating individual trees, the so-called individual tree-crown approach provides an improved methodological framework for forest characterisation. In the more common cases, delineation of individual trees is utilised by the segmentation of a canopy height model (CHM) [9, 4]. Generated by subtracting the digital terrain model (DTM) from the digital surface model (DSM), CHM provides estimations of tree heights at discrete space intervals (usually grid). Local maxima are then used to define the positions of treetops, while the segmentation is commonly achieved by applying region growing [9] or watershed algorithms [14]. Although this is an elegant approach, there are several issues related to it. Namely, due to the complex geometry and noise contained within the LiDAR data, defining treetops by local maxima alone inevitably results in oversegmentation. Typical approaches to this issue include smoothing CHM as described by [9], controlling the distances between treetops [4], and data scaling based on minimum curvature estimation as considered in [14]. Nevertheless, by assuming that individual trees form distinct peaks in the CHM, these methods tend to perform well in structurally homogeneous plantations, single-species dominated stands, and forests with widely-spaced trees. However, significantly lower accuracies are reported in more complex cases, where trees are clumped together and have well-developed under and midstorey canopy layers consisting of smaller, shade-tolerant trees.

This paper considers a new definition of markers for single tree-crown delineation. The underlying theoretical foundations are given in Section 2. Section 3 proposes a new framework for delineating single-trees. Section 4 provides the results and Section 5 concludes the paper.

2 Theoretical foundations

We define an undirected graph G by the set of vertices $V(G)$ and a set of undirected edges $E(G)$. An edge $e_{i,j} \in E(G)$ is defined by a pair of vertices $e_{i,j} = (v_i, v_j)$, where $v_i, v_j \in V(G)$. A graph $G_A \subseteq G$ is a subgraph of G if $V(G_A) \subseteq V(G)$ and $E(G_A) \subseteq E(G)$. An arbitrary vertex attribute-function, denoted as A, is a mapping $A : V(G) \to \mathbb{R}$ (e.g. spatial coordinates of vertex v_i are given as $X[v_i], Y[v_i]$, and $Z[v_i]$), while the edge weight-function is the mapping $W : E(G) \to \mathbb{R}$. An edge-weighted graph is thus given by a pair (G, W). When considering operations on weight-functions, we denote the infimum (i.e. minimum) by $\bigwedge$, and $\bigvee$ is used to denote the supremum (i.e. maximum).

Let an ordered sequence of vertices $P_{i_0,i_L} = \{v_{i_0}, v_{i_1}, \ldots, v_L\}$, we say that P_{i_0,i_L} is a path from v_{i_0} to v_{i_L} in a graph G if $\forall l \in [1, L]$ there exists an edge $e_{i_{l-1},i_l} \in E(G)$. The length of the path is given by $L = |P_{i,j}| - 1$, where notation $\hat{P}_{i,j}$ is used to emphasise that $P_{i,j}$ is the shortest path between v_i and v_j. If there exists $P_{i,j}$, we say that vertices v_i and v_j are linked and a graph G is called connected if $\forall v_i, v_j \in V(G)$ there exists $P_{i,j}$ in G. A subgraph $C_i \subseteq G$ is called a connected component of G if C_i is connected and is maximal for this property. Each C_i is uniquely addressed by any member vertex $v_i \in C_i$. When considering two non-empty subgraphs

*This work was supported by the Slovenian Research Agency under grants J2-5479, J2-6764, and P2-0041.

†Faculty of Electrical Engineering and Computer and Science, University of Maribor, Slovenia.

$G_A, G_B \subseteq G$, G_B is called an extension of G_A in G if $G_A \subseteq G_B$ and any connected component of G_B contains exactly one connected component of G_A. Consequently, a set of edges $E(G_C) \subseteq E(G)$ is a graph-cut for G_A if $\overline{E(G_C)}$ is an extension of G_A and if $E(G_C)$ is minimal for this property [2]. We denote a graph-cut for G_A as $E^C(G_A)$. Another particularly important subgraph for the following definitions is a graph of local minima. These are defined as connected subgraphs of equally weighted edges, the adjacent edges of which have strictly greater values. A graph union of local minima is denoted as $M(G, W)$.

2.1 Watershed

The definition of watershed used in this paper relies on the notion of the minimum spanning forest $MFS(G, W)$, constructed on (G, W), relative to the set of local minima $M(G, W)$. Let $G_A \subseteq G$, we say that G_A is a forest relative to $M(G, W)$ if G_A is an extension of $M(G, W)$ and for any extension G_B of $M(G, W)$, such that $G_B \subseteq G_A$, $V(G_A) = V(G_B)$ implies $G_A = G_B$. When $V(G_A) = V(G)$, we say G_A is a spanning forest relative to $M(G, W)$ for G. $MSF(G, W)$ is, thus, a spanning forest relative to $M(G, W)$, the sum of edge-weights of which is less than or equal to the sum of the edge-weights of any other spanning forest relative to $M(G, W)$ for (G, W). As shown by [2], a watershed is a graph-cut $E^C(MSF(G, W))$ for $MSF(G, W)$ or, equivalently, watershed regions are connected components $C_i \in MSF(G, W)$.

In addition to its efficient implementation (see the pseudocode in [2]), the given watershed definition allows for flexible graph segmentation by manipulating over the edge weight functions. When considering the classical watershed problem where those "valleys" separated by "ridges" are being segmented, edge-weights are defined by the lowest attribute value of the corresponding vertices, i.e. $W[e_{i,j}] = \bigwedge\{A[v_i], A[v_j]\}$. Inversely, the segmentation of peaks is achieved by $W[e_{i,j}] = \bigwedge\{-A[v_i], -A[v_j]\}$. Furthermore, a marker-controlled watershed can be achieved by filtering the underlying attribute function A, discussed next.

2.2 Connected operators on graphs

We define the neighbourhood of a vertex v_i (i.e. structuring element) by a subgraph $S_i^s \subseteq G$ of size s, the vertex-set and edge-set of which are:

$$\begin{aligned} V(S_i^s) &= \{v_j \in V(G) \ : \ |\hat{P}_{i,j}| - 1 \leq s\} \cup \{v_i\}, \\ E(S_i^s) &= \{e_{j,k} \in E(G) \ : \ v_j, v_k \in V(S_i^s)\}. \end{aligned} \quad (1)$$

In other words, vertex-set of S_i^s consists of vertices whose shortest paths from v_i are no longer than s and its edge-set contains the corresponding edges from $E(G)$. A dilation of A at v_i is then defined as $\delta^s(A)[v_i] = \bigvee_{v_j \in V(S_i^s)} A[v_j]$, while an elementary geodesic dilation $\delta_A^1(A^M)$ of a marker attribute-function A^M under the mask A is according to [13] given as

$$\delta_A^1(A^M)[v_i] = \bigwedge\{\delta^1(A)[v_i], A[v_i]\}. \quad (2)$$

Geodesic dilation of size $s \geq 1$, denoted as δ_A^s, is defined by iterating elementary geodesic dilation $s-$times. As only undirected graphs are considered in this paper, elementary erosion is given by the duality principle as $\epsilon^s(A) = -\delta^s(-A)$ and geodesic erosion as $\epsilon_A^s(A^M) = -\delta_{-A}^s(-A^M)$. A straightforward approach to marker-controlled watershed is achieved by the so-called reconstruction closing of A under the marker-function A^M [12]. For this purpose, $A^M[v_i] = A[v_i]$ if v_i is marked and $A^M[v_i] = \infty$ otherwise. Reconstruction by erosion, denoted as ϵ^R, is defined as a stable state achieved after iteratively applying elementary geodesic erosions [10, 8]. Accordingly,

$$\epsilon_A^R(A^M) = \bigwedge_{s \to \infty} \epsilon_A^s(A^M). \quad (3)$$

The edge-weight function allowing for marker-controlled watershed is then given as $W[e_{i,j}] = \bigwedge\{\epsilon_A^R(A^M)[v_i], \epsilon_A^R(A^M)[v_j]\}$ or $W[e_{i,j}] = \bigwedge\{-\epsilon_A^R(A^M)[v_i], -\epsilon_A^R(A^M)[v_j]\}$ when peaks are being segmented rather than valleys. Based on these notions, a new approach for single-tree delineation in LiDAR data is proposed next.

3 Single tree-crown delineation

During the initial step of the method, the topology construction over the input LiDAR point cloud is done in consistency with the graph-theoretical approach. Although any type of undirected graph can be used, sufficient memory and computational efficiency is achieved by constructing G as a regular 2D grid for the representation of CHM. A digital surface model is then constructed as a mapping $DSM : V(G) \to \mathbb{R}$, where $DSM[v_i]$ is defined by the highest point within the neighbourhood of v_i (i.e. the considered grid cell). DTM is in our case constructed as described by [7] and then subtracted from DSM. The heights of the vertices corresponding to building regions are set at 0 in order to obtain CHM.

3.1 Segmentation of canopy layer

This section proposes an improved treetop detection based on locally fitted surfaces (LoFS), leading to a significantly lower oversegmentation in comparison with the usually used local maxima search. Initially proposed for the extraction of planar points in [7], LoFS uses surface-fitting based on least squares in

order to obtain descriptions of vertices' local neighbourhoods. As the least-square fitting is well known, it is not discussed at this point. An in-depth study of the least-squares is provided by [1] and [11]. Let $\Pi^s(G_2, CHM)$ be a set of polynomials, best-fitted to CHM within the neighbourhood of a given vertex v_i (note that the neighbourhood of v_i is defined by a subgraph $S_i^s \subseteq G_2$). The best-fitted polynomial $\Pi^s(G_2, CHM)[v_i]$ is given in explicit form as

$$\begin{aligned}&\Pi^s(G_2, CHM)[v_i] = \\ &c_5X[v_i]^2 + c_4X[v_i]Y[v_i] + \\ &+c_3Y[v_i]^2 + c_2X[v_i] + c_1Y[v_i] + c_0.\end{aligned} \tag{4}$$

When considering coefficients c_k (where $k \in [0, 5]$) defining Π^s, the neighbourhood of v_i at scale s can be segmented into convex, concave, or neither of those, i.e. a saddle [3]. Namely, it can be shown that $\Pi^s(G_2, CHM)[v_i]$ is **saddle** if $c_4^2 - 4c_5c_3 \geq 0$. Otherwise, the neighbourhood is **convex** if $c_5 \geq 0$ and **concave** if $c_5 < 0$.

Intuitively, vertices with concave neighbourhoods represent treetops, while saddle and convex neighbourhoods appear near the boundaries of tree-crowns. As the used surface approximation is, by itself, capable of absorbing a fair amount of noise, treetop detection is significantly more reliable in this way. It is also better adapted for dealing with anisotropic tree-crowns than e.g. Gaussian filtering at variable scales [5]. Most importantly, the proposed approach allows for controlling the curvature of the treetops by applying a threshold $\check{t}_M$ instead of using a constant 0. Thus, a marker function CHM_M used for marker-controlled watershed is given as

$$CHM_M[v_i] = \begin{cases} CHM[v_i], & \text{if concave.} \\ \infty, & \text{otherwise.} \end{cases} \tag{5}$$

The edge-weight function, allowing for watershed segmentation based on LoFS, is defined as

$$\begin{aligned}&W_\Pi[e_{i,j}] = \\ &\bigwedge\{-\epsilon^R_{CHM}(CHM_M)[v_i], -\epsilon^R_{CHM}(CHM_M)[v_i]\}.\end{aligned} \tag{6}$$

Watershed is thus obtained by a cut $E^C(MSF(G_2, W_\Pi))$ and watershed regions defining individual trees are connected components $C_i \in MSF(G_2, W_\Pi)$.

4 Results

The proposed approach was evaluated on six different LiDAR datasets of different terrain configuration, forest types, and data density. Within these datasets, 11 test areas were selected where ground truth measurements were acquired. The centres of each test area were defined first and the positions of the trees within a radius of $12m$ were measured. In total, the positions of 232 trees were sampled using *Garmin GPSMAP 60CSx*. The estimated average error was approximately $2m$ in all the test cases, with standard deviation equal to $0.5m$.

In the first step of the validation procedure, detected trees within the test areas are extracted first by thresholding them based on their distances from the closest centres of the test areas. These are then matched by linking them with the closest ground-truth positions of the tree trunks, where: i) *tp* are true positives, defined by the number of ground truth trees that were detected, ii) *fp* are false positives, detected if a ground truth tree is matched with more than one estimate, and iii) *fn* are false negatives, defined by the number of ground truth trees that were not matched. Accordingly, the following statistical measurements of the performance were used [6]: **i) sensitivity** or true positive rate *tpr* is given by $tpr = tp/(tp+fn)$, **ii) precision** or positive predictive value *ppv* is defined as $ppv = tp/(tp + fp)$, and **iii)** $f1$**-*score*** is the harmonic mean of precision and sensitivity, given by $f1\text{-}score = 2tp/(2tp + fp + fn)$.

4.1 Validation

The proposed approach for the delineation of *CHM* based on *LoFS* was evaluated in comparison to the traditional approach with local maxima-based definition of watershed markers. As the compared method relies on smoothing in order to cope with oversegmentation, different smoothing levels (i.e $\sigma \in \{0.5, 0.75, 1.0, 1.25, 1.5\}$) were applied. On average, the highest $f1\text{-}score$ value was achieved when $\sigma \doteq 0.75$ and the values of *tpr* and *ppv* were balanced (see Fig. 1a) as σ positively correlates with *ppv*, while its correlation with *tpr* is negative. Although only the highest $f1\text{-}score$ achieved by the compared method was used in further comparisons, significant improvement in the results was achieved with the proposed *LoFS*-based definition of markers, where curvature threshold $\check{t}_M = 0.01$ was used in all the cases (see Eq. 5).

The proposed method achieved lower *tpr* value in comparison to the traditional approach (see Fig. 1), however, its precision given by *ppv* was greater. The decreased *tpr* can be attributed to the inaccuracies of the ground-truth data and the matching procedure that may coincidentally match the overdetected trees from the canopy layer with the understorey trees. Nevertheless, the proposed method achieved higher $f1\text{-}score$ in all cases. This indicated that the proposed method was better adapted for dealing with noise and confirmed that the *LoFS*-based definition of markers outperformed the local maxima search as it is better adapted to dealing with the anisotropic shapes of tree-crowns of different sizes than straightforward smoothing of *CHMs* at various levels.

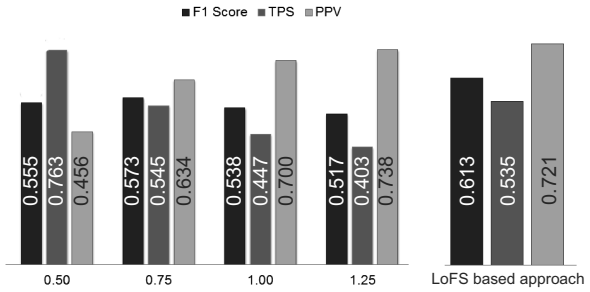


Figure 1: The accuracy of single tree-crown extraction of the compared method at different smoothing levels (left) and the increase in efficiency achieved by the proposed method (right).

5 Conclusion

This paper proposes an improved CHM analysis for single tree crown delineation using *LoFS*-based definitions of markers. Since *LoFS* can be asymmetrical and of arbitrary curvature, the proposed approach is superior to the traditional predifined filters, e.g. Gaussian kernel. As confirmed by the results, this self-adaptation to various sizes and anisotropic shapes of tree-crowns results in approximately 10% improvement of extraction rate, while the precision is decreased by less than 3%.

Acknowledgments

A special thanks goes to the Slovenia Forest Service for providing us with LiDAR and ground truth data. This paper was produced within the framework of the operation entitled "Centre of Open innovation and Research UM". The operation is co-funded by the European Regional Development Fund and conducted within the framework of the Operational Programme for Strengthening Regional Development Potentials for the period 2007-2013, development priority 1: "Competitiveness of companies and research excellence", priority axis 1.1: "Encouraging competitive potential of enterprises and research excellence".

References

[1] P. Bevington and D. K. Robinson. *Data Reduction and Error Analysis for the Physical Sciences*. McGraw-Hill, New York, 3th edition, 2002.

[2] J. Cousty, G. Bertrand, L. Najman, and M. Couprie. Watershed cuts: Minimum spanning forests and the drop of water principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(8):1362–1374, 2009.

[3] I. S. Gradshteyn and I. M. Ryzhik. *Tables of Integrals, Series, and Products*. Academic Press, San Diego, CA, 6th edition, 2000.

[4] W. Li, Q. Guo, M. K. Jakubowski, and M. Kelly. A new method for segmenting individual trees from the LiDAR point cloud. *Photogrammetric Engineering & Remote Sensing*, 78(1):75–84, 2012.

[5] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[6] P. S. Mann. *Introductory Statistics (7th ed.)*. John Wiley & Sons Inc., 2010.

[7] D. Mongus, N. Lukač, and B. Žalik. Ground and building extraction from LiDAR data based on differential morphological profiles and locally fitted surfaces. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:145–156, 2014.

[8] L. Najman and H. Talbot. *Mathematical Morphology*. John Wiley & Sons, 2013.

[9] Å. Persson, J. Holmgren, and U. Söderman. Detecting and measuring individual trees using an airborne laser scanner. *Photogrammetric Engineering and Remote Sensing*, 68(9):925–932, 2002.

[10] P. Salembier and M. H. F. Wilkinson. Connected operators: A review of region-based morphological image processing techniques. *IEEE Signal Processing Magazine*, 136(6):136–157, 2009.

[11] P. Schneider and D. H. Eberly. *Geometric tools for computer graphics*. Morgan Kaufmann, 2002.

[12] C. Vachier and F. Meyer. The viscous watershed transform. *Journal of Mathematical Imaging and Vision*, 22(2-3):251–267, 2005.

[13] L. Vincent. Morphological gray scale reconstruction in image analysis: Applications and efficients algorithms. *IEEE Transactions on Image Processing*, 2(2):76–201, 1993.

[14] X. Yu, J. Hyyppä, M. Vastaranta, M. Holopainen, and R. Viitala. Predicting individual tree attributes from airborne laser point clouds based on the random forests technique. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(1):28–37, 2011.

Low-quality dimension reduction and high-dimensional Approximate Nearest Neighbor *

Evangelos Anagnostopoulos† Ioannis Z. Emiris‡ Ioannis Psarros†

Abstract

We generalize the Johnson-Lindenstrauss lemma to define "low-quality" mappings to a Euclidean space of significantly lower dimension which allow us to solve the approximate nearest neighbor problem (ϵ-ANN). With high probability, an approximate nearest neighbor lies among the k approximate nearest neighbors in the projected space. In order to compute them, we employ a data structure, such as BBD-trees. Our algorithm, given n points in $\mathbb{R}^d$, achieves space usage in $O(dn)$, preprocessing time in $O(dn \log n)$, and query time in $O(dn^\rho \log n)$, where ρ is proportional to $1 - 1/\ln \ln n$, for fixed $\epsilon \in (0,1)$. The dimension reduction is larger if one assumes that pointsets possess some structure, namely bounded expansion rate. We present experimental results in up to 500 dimensions and 10^5 points.

1 Introduction

Nearest neighbor searching is a fundamental computational problem. Let X be a set of n points in $\mathbb{R}^d$ and let $d(p,p')$ be the (Euclidean) distance between any two points p and p'. The problem consists in building a data structure which reports, given a query point q, a point $p \in X$ s.t. $d(p,q) \leq d(p',q)$, for all $p' \in X$ and p is said to be a nearest neighbor of q. However, an exact solution to high-dimensional nearest neighbor search, in sublinear time, requires prohibitively heavy resources. Thus, many techniques focus on the less demanding task of computing the approximate nearest neighbor (ϵ-ANN). Given a parameter $\epsilon \in (0,1)$, a $(1+\epsilon)$-approximate nearest neighbor to a query q is a point p in X s.t. $d(q,p) \leq (1+\epsilon) \cdot d(q,p')$, $\forall p' \in X$.

Sec. 2 introduces our embeddings to dimension lower than predicted by the Johnson-Linderstrauss lemma. Sec. 3 states our main results about ϵ-ANN search. Sec. 4 generalizes our discussion so as to exploit bounded expansion rate, and Sec. 5 presents experiments. This extended abstract reformulates and summarizes results from [AEP15] where missing details and omitted proofs can be found.

In [AMN+98] they introduced the Balanced Box-Decomposition (BBD) trees. BBD-trees achieve query time $O(c \log n)$ with $c \leq d/2\lceil 1 + 6d/\epsilon \rceil^d$, using space in $O(dn)$, and preprocessing time in $O(dn \log n)$. An approximation to the $k \geq 1$ nearest neighbors can be computed at an extra cost of $O(k \cdot d \log n)$. Another data structure is the Approximate Voronoi Diagrams which are shown to establish a space/time tradeoff [AMM09]. For high dimensions, one might apply the Johnson-Lindenstrauss lemma and map the points to $O(\frac{\log n}{\epsilon^2})$ dimensions which however requires prohibitive resources when $\epsilon \ll 1$.

In high dimensional spaces, space partitioning data structures are affected by the curse of dimensionality. An important method conceived for high dimensional data is locality sensitive hashing (LSH). In general, LSH requires roughly $O(dn^{1+\rho})$ space and $O(dn^\rho)$ query time for some parameter $\rho \in (0,1)$. Lately, it was shown that one can achieve $\rho \leq \frac{7}{8(1+\epsilon)^2} + O(\frac{1}{(1+\epsilon)^3}) + o(1)$ [AINR14]. For comparison, in Thm. 6 we show that it is possible to use $O(dn)$ space, with query time roughly $O(dn^\rho)$ where $\rho < 1$ is now higher than the one appearing in LSH.

In [KR02], they introduce the notion of expansion rate c for an arbitrary metric which is more restrictive than that of the doubling dimension (See Sec. 4 for definitions). In Thm. 8, we provide a data structure for the ϵ-ANN problem in the Euclidean metric with $O(dn)$ space and $O((C^{1/\epsilon^3} + \log n) d \log n/\epsilon^2)$ query time, where C depends on c.

2 Low Quality Randomized Embeddings

In the following, we denote by $\|\cdot\|$ the Euclidean norm and by $|\cdot|$ the cardinality of a set. The Johnson and Lindenstrauss lemma states that for any set $X \subset \mathbb{R}^d$, $\epsilon \in (0,1)$ there exists a distribution over linear mappings $f : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}$, where $d' = O(\log |X|/\epsilon^2)$, s.t. $\forall p, q \in X$, $(1-\epsilon)\|p-q\|^2 \leq \|f(p) - f(q)\|^2 \leq (1+\epsilon)\|p-q\|^2$.

In the initial proof [JL84], they show that this can be achieved by orthogonally projecting the pointset on a random linear subspace of dimension d'. In [DG02], they provide a proof based on elementary probabilistic techniques. In [IM98], they prove that it suffices to

*This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS UOA (MIS 375891).

†Dept. of Mathematics, University of Athens, Greece, aneva@math.uoa.gr, i.psarros@di.uoa.gr

‡Dept. of Informatics & Telecommunications, University of Athens, Greece, emiris@di.uoa.gr

apply a gaussian matrix on the pointset, or even simpler a matrix whose entries are independent random variables with uniformly distributed values in $\{-1, 1\}$. However, it has been realized [IN07] that this notion of randomized embedding is stronger than what is required for the ϵ-ANN problem.

Definition 1 [IN07] *Let (Y, d_Y), (Z, d_Z) be metric spaces and $X \subseteq Y$. A distribution over mappings $f : Y \to Z$ is a* nearest-neighbor preserving embedding *with distortion $D \geq 1$ and probability of success $P \in [0, 1]$, if for $\epsilon > 0$ and $\forall q \in Y$, with probability at least P, when $x \in X$ is such that $f(x)$ is an $(1 + \epsilon)-$approximate nearest neighbor of $f(q)$ in $f(X)$, then x is a $(D \cdot (1 + \epsilon))-$approximate nearest neighbor of q in X.*

Let X be a set of n points in $\mathbb{R}^d$, let $q \in \mathbb{R}^d$ and $1 \leq k \leq n$. The problem of computing an approximation of the k nearest points (ϵ-kANNs) consists in reporting a sequence $S = \{p_1, \cdots, p_k\}$ of k distinct points s.t. the i-th point is an $(1 + \epsilon)$ approximation to the i-th nearest neighbor of q. Furthermore, Asmp. 1 is satisfied by BBD-trees.

Assumption 1 *Let $S' \subseteq X$ be the set of points visited by the ϵ-kANNs search s.t. $S = \{p_1, \cdots, p_k\} \subseteq S'$ is the set of points which are the k nearest to the query point q among the points in S'. We assume that $\forall x \in X \setminus S'$, $d(x, q) > d(p_k, q)/(1 + \epsilon)$.*

Assuming the existence of a data structure which solves ϵ-kANNs, we can weaken Def. 1 as follows.

Definition 2 *Let (Y, d_Y), (Z, d_Z) be metric spaces and $X \subseteq Y$. A distribution over mappings $f : Y \to Z$ is a* locality preserving embedding *with distortion $D \geq 1$, probability of success $P \in [0, 1]$ and locality parameter k, if for $\epsilon > 0$ and $\forall q \in Y$, with probability at least P, when $S = \{f(p_1), \cdots, f(p_k)\}$ is a solution to ϵ-kANNs for q under Asmp. 1, then $\exists f(x) \in S$ such that x is a $D \cdot (1 + \epsilon)-$approximate nearest neighbor of q in X.*

Hence, we reduce the problem of ϵ-ANN in dimension d to the problem of ϵ-kANNs in dimension $d' < d$.

Lemma 1 [DG02] *There exists a distribution over linear maps $A : \mathbb{R}^d \to \mathbb{R}^{d'}$ s.t., for any $p \in \mathbb{R}^d$ with $\|p\| = 1$:*

- *if $\beta < 1$ then $\Pr[\|Ap\|^2 \leq \beta^2 \cdot \frac{d'}{d}] \leq exp(\frac{d'}{2}(1 - \beta^2 + 2\ln\beta)$,*
- *if $\beta > 1$ then $\Pr[\|Ap\|^2 \geq \beta^2 \cdot \frac{d'}{d}] \leq exp(\frac{d'}{2}(1 - \beta^2 + 2\ln\beta)$.*

The following lemma is proved by induction [AEP15].

Lemma 2 $\forall i \in \mathbb{N}$, $\epsilon \in (0, 1)$,

$$\frac{(1 + \epsilon/2)^2}{(2^i(1 + \epsilon))^2} - 2\ln\frac{(1 + \epsilon/2)}{2^i(1 + \epsilon)} - 1 > 0.05(i + 1)\epsilon^2.$$

Lemma 3 *A simple calculation shows that $\forall x > 0$,*

$$\frac{(1 + x)^2}{(1 + 2x)^2} - 2\ln(\frac{1 + x}{1 + 2x}) - 1 < (1+x)^2 - 2\ln(1+x) - 1.$$

Theorem 4 *Under the notation of Def. 2, there exists a randomized mapping $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ which satisfies Def. 2 for $d' = O(\log\frac{n}{\delta k}/\epsilon^2)$, $D = 1 + \epsilon$ and $P = 1 - \delta$, for any constant $\delta \in (0, 1)$.*

Proof. Let X be a set of n points in $\mathbb{R}^d$ and consider map $f : \mathbb{R}^d \to \mathbb{R}^{d'} : v \mapsto \sqrt{d/d'} \cdot A\ v$, where A is a matrix as in Def. 2. Wlog the query point q lies in the origin and its nearest neighbor u lies at distance 1 from q. We denote by $c \geq 1$ the approximation ratio guaranteed by the assumed data structure. In other words, the assumed data structure solves the $(c - 1)$-kANNs problem.

For each point x, $L_x = \|Ay\|^2$ where $y = x/\|x\|$. Let N be the random variable whose value indicates the number of "bad" candidates, that is

$$N = |\{x \in X : \|x - q\| > \gamma \wedge L_x \leq \frac{\beta^2}{\gamma^2} \cdot \frac{d'}{d}\}|,$$

where we define $\beta = c(1 + \epsilon/2)$, $\gamma = c(1 + \epsilon)$. Hence, by Lem. 1, $\mathbb{E}[N] \leq n \cdot exp(\frac{d'}{2}(1 - \frac{\beta^2}{\gamma^2} + 2\ln\frac{\beta}{\gamma}))$. By Markov's inequality,

$$\Pr[N \geq k] \leq \frac{\mathbb{E}[N]}{k} \leq \frac{n}{k} \cdot exp(\frac{d'}{2}(1 - \frac{\beta^2}{\gamma^2} + 2\ln\frac{\beta}{\gamma})).$$

The event of failure is defined as the disjunction of two events: $[N \geq k] \vee [L_u \geq (\beta/c)^2 d'/d]$, and its probability is at most equal to

$$\Pr[N \geq k] + exp(\frac{d'}{2}(1 - (\beta/c)^2 + 2\ln(\beta/c))),$$

by applying again Lem. 1. Now, we bound these two terms. For the first we choose d' s.t.

$$d' \geq 2\,\frac{\ln\frac{2n}{\delta k}}{\frac{\beta^2}{\gamma^2} - 1 - 2\ln\frac{\beta}{\gamma}}. \tag{1}$$

Therefore,

$$\frac{exp(\frac{d'}{2}(1 - \frac{\beta^2}{\gamma^2} + 2\ln\frac{\beta}{\gamma}))}{k} \leq \frac{\delta}{2n} \implies \Pr[N \geq k] \leq \frac{\delta}{2}. \tag{2}$$

Notice that $k \leq n$ and due to Lem. 2, we obtain $(\beta/\gamma)^2 - 2\ln(\beta/\gamma) - 1 < (\beta/c)^2 - 2\ln(\beta/c) - 1$. Hence, inequality (1) implies $d' \geq 2\frac{\ln\frac{2}{\delta}}{(\beta/c)^2 - 1 - 2\ln(\beta/c)} \implies$

$$\implies exp(\frac{d'}{2}(1 - (\beta/c)^2 + 2\ln(\beta/c))) \leq \frac{\delta}{2}. \tag{3}$$

Inequalities (2), (3) imply that the total probability of failure is at most δ. By Lem. 2 there exists $d' =$

$O(\log \frac{n}{\delta k}/\epsilon^2)$ and with probability of success at least $1-\delta$, $\|f(q)-f(u)\| \leq (1+\frac{\epsilon}{2})\|u-q\|$ and $N < k$.

Now assume success and let $S = \{f(p_1), ..., f(p_k)\}$ a solution of the $(c-1)$-kANNs problem in $f(X)$, satisfying Asmp. 1. We have that $\forall f(x) \in f(X) \setminus S'$, $\|f(x)-f(q)\| > \|f(p_k)-f(q)\|/c$ where S' is the set of all points visited by the search routine.

Now, if $f(u) \in S$ then S contains the projection of the nearest neighbor. If $f(u) \notin S$ then if $f(u) \notin S'$,

$$\|f(p_k)-f(q)\| < c\|f(u)-f(q)\| \leq c(1+\epsilon/2)\|u-q\|.$$

Hence, $\exists f(p^*) \in S$ s.t. $\|q-p^*\| \leq c(1+\epsilon)\|u-q\|$. Finally, if $f(u) \notin S$ but $f(u) \in S'$,

$$\|f(p_k)-f(q)\| \leq \|f(u)-f(q)\| \leq (1+\epsilon/2)\|u-q\|$$

and $\exists f(p^*) \in S$ s.t. $\|q-p^*\| \leq c(1+\epsilon)\|u-q\|$. □

3 Approximate Nearest Neighbor Search

Notice, that BBD-trees satisfy the Asmp. 1. The ϵ-kANNs search, visits cells in increasing order with respect to their distance from the query and it stops when the current cell lies in distance $> r_k/c$ where r_k is the current distance to the kth nearest neighbor. We set $D = 1+\epsilon$, thus relating approximation error to the allowed distortion; this is not required but simplifies the analysis. Our analysis then focuses on the asymptotic behaviour of the term $\lceil 1+6\frac{d'}{\epsilon}\rceil^{d'}+k$.

Lemma 5 *With the above notation, there exists $k > 0$ s.t., for fixed $\epsilon \in (0,1)$, it holds that $\lceil 1+6\frac{d'}{\epsilon}\rceil^{d'}+k = O(n^\rho)$, where $\rho \leq 1-\epsilon^2/\hat{c}(\epsilon^2+\ln(\max(\frac{1}{\epsilon},\ln n))) < 1$ for some constant $\hat{c} > 1$.*

Proof. Recall that $d' \leq \frac{\tilde{c}}{\epsilon^2}\ln\frac{n}{k}$ for some constant $\tilde{c} > 0$. The constant δ is hidden in $\tilde{c}$. Let $k = n^\rho$. Obviously $\lceil 1+6\frac{d'}{\epsilon}\rceil^{d'} \leq (c'\frac{d'}{\epsilon})^{d'}$, for some $c' \in (1,7)$.

$$(c'\frac{d'}{\epsilon})^{d'} = n^{\frac{\tilde{c}(1-\rho)}{\epsilon^2}\ln(\frac{\tilde{c}c'(1-\rho)\ln n}{\epsilon^3})}. \quad (4)$$

We assume $\epsilon \in (0,1)$ is a fixed constant. Hence, it is reasonable to assume that $\frac{1}{\epsilon} < n$. We consider two cases when comparing $\ln n$ to ϵ:

- $\frac{1}{\epsilon} \leq \ln n$. Substituting $\rho = 1-\frac{\epsilon^2}{2\tilde{c}(\epsilon^2+\ln(c'\ln n))}$ into equation (4), the exponent of n is bounded as follows: $\frac{\tilde{c}(1-\rho)}{\epsilon^2}\ln(\frac{\tilde{c}c'(1-\rho)\ln n}{\epsilon^3}) < \rho$.
- $\frac{1}{\epsilon} > \ln n$. Substituting $\rho = 1-\frac{\epsilon^2}{2\tilde{c}(\epsilon^2+\ln\frac{c'}{\epsilon})}$ into equation (4), the exponent of n is bounded by ρ.

□

Combining Thm. 4 with Lem. 5 yields Thm. 6.

Theorem 6 (Main) *Given n points in $\mathbb{R}^d$, there exists a randomized data structure which requires $O(dn)$ space and reports an $(1+\epsilon)^2$-approximate nearest neighbor in time $O(dn^\rho \log n)$, where $\rho \leq 1-\epsilon^2/\hat{c}(\epsilon^2+\ln(\max(\frac{1}{\epsilon},\ln n)))$ for some constant $\hat{c} > 1$. The preprocessing time is $O(dn\log n)$. For each query, preprocessing succeeds with constant probability.*

Proof. The size of the input dataset is $O(dn)$. The space used by BBD-trees is $O(d'n)$ and we also need $O(dd')$ space for the matrix A. Hence, since $d' < d$ and $d' < n$, the total space usage is $O(dn)$.

Building the BBD-tree costs $O(d'n\log n)$ time. We sample a d'-dimensional random subspace as follows. We sample in time $O(dd')$, a $d \times d'$ matrix whose entries are independent random variables with the standard normal distribution. Then, we orthonormalize with Gram-Schmidt in time $O(dd'^2)$. Since $d' = O(\log n)$, the total preprocessing time is $O(dn\log n)$.

We need $O(dd')$ time to project each query. Next, we compute its ϵ-n^ρANNs in time $O(d'n^\rho\log n)$ and we check them with their real coordinates in time $O(dn^\rho)$. Hence, each query costs $O(dn^\rho\log n)$. □

4 Bounded Expansion Rate

This section models the structure that the data may have so as to obtain more precise bounds. To this end, we consider pointsets with bounded expansion rate.

Definition 3 *Let M a metric space and $X \subseteq M$ a finite pointset and let $B_p(r)$ denote the points of X lying in the ball centered at p with radius r. We say that X has (ρ,c)-expansion rate if and only if, $\forall p \in M$ and $r > 0$, $|B_p(r)| \geq \rho \implies |B_p(2r)| \leq c\cdot|B_p(r)|$.*

Theorem 7 *There exists a randomized mapping $f: \mathbb{R}^d \to \mathbb{R}^{d'}$ which satisfies Def. 2 for $d' = O(\frac{\log(c+\frac{\rho}{\delta k})}{\epsilon^2})$, $D = 1+\epsilon$ and $P = 1-\delta$, for any constant $\delta \in (0,1)$, for pointsets with (ρ,c)-expansion rate.*

Proof. (Sketch) We proceed in the same spirit as in the proof of Thm. 4, and using the notation from that proof. Let r_0 be the distance to the ρ−th nearest neighbor, excluding neighbors at distance $\leq 1+\epsilon$. For $i > 0$, let $r_i = 2\cdot r_{i-1}$ and set $r_{-1} = 1+\epsilon$. $\mathbb{E}[N] \leq$

$$\sum_{i=0}^{\infty} c^i\rho\cdot exp(\frac{d'}{2}(1-\frac{(1+\epsilon/2)^2}{2^{2i}(1+\epsilon)^2}+2\ln\frac{1+\epsilon/2}{2^i(1+\epsilon)})).$$

By Lem. 2 and for $d' \geq 40\cdot\ln(c+\frac{2\rho}{k\delta})/\epsilon^2$, $\mathbb{E}[N] \leq$

$$\rho\sum_{i=0}^{\infty} c^i\cdot(\frac{1}{c})^{i+1}\cdot(\frac{1}{1+\frac{2\rho}{kc\delta}})^{i+1} = \frac{\rho}{c}\sum_{i=0}^{\infty}(\frac{1}{1+\frac{2\rho}{kc\delta}})^{i+1} = \frac{k\delta}{2}.$$

Finally, $\Pr[N \geq k] \leq \frac{\mathbb{E}[N]}{k} \leq \frac{\delta}{2}$. □

Employing Thm. 7 we obtain a result analogous to Thm. 6 which underlines our scheme's sensitivity to real world assumptions.

Theorem 8 *Given n points in $\mathbb{R}^d$ with $(\ln n, c)$-expansion rate, there exists a randomized data structure which requires $O(dn)$ space and $O(dn \log n)$ preprocessing time and reports an $(1+\epsilon)^2$-approximate nearest neighbor in time $O((C^{1/\epsilon^3} + \log n)d \log n/\epsilon^2)$, for some C depending on c. For each query, preprocessing succeeds with constant probability.*

5 Experiments

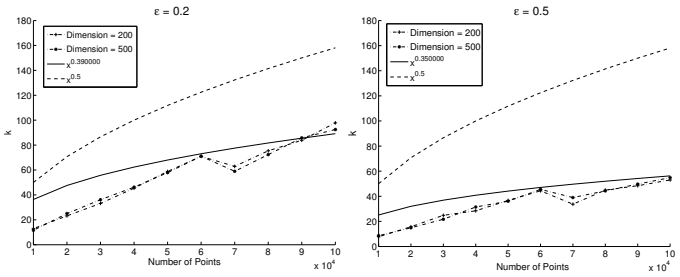


Figure 1: Plot of k as n increases. We also show functions of the form $n^\rho, \rho < 1$ for comparison.

We follow the "planted nearest neighbor model" procedure for our datasets, which is described in [DI04]. This model, which is motivated by real applications, guarantees for each query q the existence of a few approximate nearest neighbors while keeping all other points sufficiently far from q. These datasets constitute a worst-case input for our approach since every query point has exactly one nearest neighbor and has many points lying near the boundary of $(1+\epsilon)$. Following the above scheme we create datasets for different values of n, d, ϵ and for each query point we plant one neighbor at distance $R = 2$ whereas all other points lie at distance $> (1+\epsilon)R$.

We use a random mapping $f : \mathbb{R}^d \mapsto \mathbb{R}^{d'}$ where $d' = \frac{\log n}{\log \log n}$ using a gaussian matrix G, where each entry $G_{ij} \sim N(0,1)$. Then, we count the number of points lying between $f(q)$ and its actual nearest neighbor which is the optimal k for q. In Figure 1 we present the average value of k for increasing number of points n. The resulting curve has an intrinsic concavity, which shows that the dependency of k on n is sublinear. While choosing k as in Thm. 4 is quite pessimistic, it is an open question whether there exists a method for choosing k depending on the pointset.

Next, we build a BBD-tree on the projected data points using the `ANN` library. Having fixed $k = \sqrt{n}$, we measure the average time needed, for each query q, to find its ϵ-kANNs in the projected space and then to find the point which lies in distance $\leq R$. We compare these times to the average times reported by the E2LSH R-near neighbor queries. We require from both approaches to have accuracy > 0.90. It is clear from Figure 2 that E2LSH is faster than our approach by a factor of 3, however it requires more space.

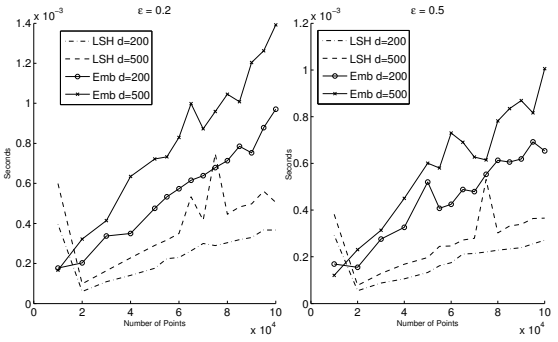


Figure 2: Our embedding approach against E2LSH.

References

[AEP15] E. Anagnostopoulos, I. Z. Emiris and I. Psarros, Low-quality dimension reduction and high-dimensional Approximate Nearest Neighbor. In arXiv:1412.1683, to appear in *SoCG* 2015.

[AINR14] A. Andoni, P. Indyk, H.L. Nguy En and I. Razenshteyn, Beyond Locality-Sensitive Hashing. In *Proc. SODA*, 2014.

[AMM09] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57(1):1–54, 2009.

[AMN+98] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman and A.Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *J. ACM*, 45(6):891–923, 1998.

[DG02] S. Dasgupta and A. Gupta, An Elementary Proof of a Theorem of Johnson and Lindenstrauss, In *Random Struct. Algorithms*, 22:60–65, 2003.

[DI04] M. Datar, N. Immorlica, P. Indyk and V. S. Mirrokni Locality-sensitive hashing scheme based on p-stable distributions, In *Proc. SoCG*, 253–262, 2004.

[IM98] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality, In *Proc. STOC*, 604–613, 1998.

[IN07] P. Indyk and A. Naor. Nearest-neighbor-preserving embeddings. *ACM Trans. Algorithms*, 3(3), 2007.

[JL84] W. B. Johnson and J. Lindenstrauss, Extensions of Lipschitz mappings into a Hilbert space, *Contemp. Math* 26 (1984), 189–206.

[KR02] D. R. Karger and M. Ruhl, Finding Nearest Neighbors in Growth-restricted Metrics, In *Proc. STOC*, 741–750, 2002.

Time-Space Trade-offs for Voronoi Diagrams

Matias Korman* Wolfgang Mulzer† André van Renssen* Marcel Roeloffzen‡ Paul Seiferth†
Yannik Stein†

Abstract

Let S be a planar n-point set. Classically, one can find the Voronoi diagram $\mathrm{VD}(S)$ for S in $O(n \log n)$ time and $O(n)$ space. We study the situation when the available workspace is limited: for $s \in \{1, \ldots, n\}$, an s-workspace algorithm has read-only access to an input array with the points from S in arbitrary order, and it may use only $O(s)$ additional words of $\Theta(\log n)$ bits for reading and writing intermediate data. We describe a randomized s-workspace algorithm for finding $\mathrm{VD}(S)$ in expected time $O((n^2/s) \log s + n \log s \log^* s)$. This almost matches the optimal running times for both constant and linear workspace and provides a continuous trade-off between them.

1 Introduction

Since the beginning of computer science, there has been interest in algorithms that can deal with strong memory constraints. This started in the early 70's [10] when memory was expensive, but now it is motivated by the proliferation of small embedded devices where a lot of memory is neither feasible nor desirable.

Even when space is not an issue, one might want to limit the number of write operations: even though one can read flash memory very fast, writing (or even reordering) data is slow and reduces the lifetime; write-access to removable memory is sometimes limited for technical or security reasons; similar problems occur when concurrent algorithms need to access the same data and create concurrency problems. A way to deal with this is to consider algorithms that do not modify the input, and use as few variables as possible.

The exact setting may vary, but there is a common theme: the input resides in a read-only data structure, the output must be written to some write-only structure, and we can use $O(s)$ additional variables to compute the solution (for a given parameter s). Our aim is to design algorithms whose running time decreases as s grows, giving a *time-space trade-off* [11].

One of the initial problems considered in this model is *sorting* [7, 8]. It is known that the time-space product of any sorting algorithm is $\Omega(n^2)$ [5], and matching upper bounds for the case $b \in \Omega(\log n) \cap O(n/\log n)$ were obtained by Pagter and Rauhe [9] (b denotes the size of the workspace, in bits). Since the sorted list cannot be stored explicitly in memory, we must report the values one by one in order.

The concept of memory constrained algorithms was introduced to computational geometry by Asano et al. [2]. They show how to compute many classic geometric structures with $O(1)$ workspace. Afterwards, several time-space trade-off algorithms have been designed for classic problems within a simple polygon, such as shortest path computation [1], visibility [4], or computing the convex hull of a simple polygon [3].

Problem Setting. We are given a list S of n points in the plane. We assume that the points are in some structure (say, an array) that allows random access to any point. We would like to design an algorithm that computes the Voronoi diagram of S, $\mathrm{VD}(S)$. Since the diagram itself cannot be explicitly stored in memory, the aim is to report its vertices one by one in a write-only data structure in no particular order. In addition to the input, the algorithm can use $O(s)$ variables (for some parameter $s \leq n$). We assume that each variable or pointer contains a data word of $\Theta(\log n)$ bits.

Our aim is an algorithm whose running time decreases as s grows. Ideally, we would like that the trade-off is continuous and that the running times for both extremes of s match with the currently best known algorithms for these cases ($O(n^2)$ and $O(n \log n)$ time for $s = 1$ and $s = n$, respectively). As we will see below, we can almost achieve this goal.

2 Voronoi Diagrams Through Random Sampling

Given a planar n-point set S, we would like to find the vertices of $\mathrm{VD}(S)$. Let $K = \{p_1, p_2, p_3\}$ be a triangle with $S \subseteq \mathrm{conv}(K)$ so that all vertices of $\mathrm{VD}(S)$ are vertices of $\mathrm{VD}(S \cup K)$. We use random sampling to divide the problem of computing $\mathrm{VD}(S \cup K)$ into $O(s)$ subproblems of size $O(n/s)$. First, we show how to take a random sample from S with small workspace.

Lemma 1 *We can sample a uniform random subset $R \subseteq S$ of size s in time $O(n + s \log s)$ and space $O(s)$.*

*National Institute of Informatics (NII), Tokyo, Japan; JST, ERATO, Kawarabayashi Large Graph Project. {korman, andre}@nii.ac.jp

†Institut für Informatik, Freie Universität Berlin, Germany. {mulzer,pseiferth,yannikstein}@inf.fu-berlin.de

‡Tohoku University, Japan. marcel@dais.is.tohoku.ac.jp

Proof. We sample a random sequence I of s distinct numbers from $\{1,\dots,n\}$. This is done in s *rounds*. At the beginning of round k, for $k=1,\dots,s$, we have a sequence I of $k-1$ numbers from $\{1,\dots,n\}$. We store I in a binary search tree T. We maintain the invariant that each node in T with value in $\{1,\dots,n-k+1\}$ stores a pointer to a unique number in $\{n-k+2,\dots,n\}$ that is not in I. In round k, we sample a random number x from $\{1,\dots,n-k+1\}$, and we check in T whether $x\in I$. If not, we add x to I. Otherwise, we add to I the number that x points to. Let y be the new element. We add y to T. Then we update the pointers: if $x=n-k+1$, we do nothing. Now suppose $x<n-k+1$. Then, if $n-k+1\notin I$, we put a pointer from x to $n-k+1$. Otherwise, if $n-k+1\in I$, we let x point to the element that $n-k+1$ points to. This keeps the invariant and takes $O(\log s)$ time and $O(s)$ space. We continue for s rounds. Any sequence of s distinct numbers in $\{1,\dots,n\}$ is sampled with equal probability.

Finally, we scan through S to obtain the elements whose positions correspond to the numbers in I. This requires $O(n)$ time and $O(s)$ space. □

We use Lemma 1 to find a random sample $R\subseteq S$ of size s. We compute $\mathrm{VD}(R\cup K)$, triangulate the bounded cells and construct a planar point location structure for the triangulation. This takes $O(s\log s)$ time and $O(s)$ space. Given a vertex $v\in\mathrm{VD}(R\cup K)$, the *conflict circle* of v is the largest circle with center v and no point from $R\cup K$ in its interior. The *conflict set* B_v of v contains all points from S that lie in the conflict circle of v, and the *conflict size* b_v of v is the number of points in B_v. We scan through S to find the conflict size b_v for each vertex $v\in\mathrm{VD}(R\cup K)$: every Voronoi vertex has a counter that is initially 0. For each $p\in S\setminus(R\cup K)$, we use the point location structure to find the triangle Δ of $\mathrm{VD}(R\cup K)$ that contains it. At least one vertex v of Δ is in conflict with p. Starting from v, we walk along the edges of $\mathrm{VD}(R\cup K)$ to find all Voronoi vertices in conflict with p. We increment the counters of all these vertices. This may take a long time in the worst case, so we impose an upper bound on the total work. For this, we choose a *threshold* M. When the sum of the conflict counters exceeds M, we start over with a new sample R. The total time for one attempt is $O(n\log s+M)$, and below we prove that for $M=\Theta(n)$ the success probability is at least $3/4$. Next, we pick another threshold T, and we compute for each vertex v of $\mathrm{VD}(R\cup K)$ the *excess* $t_v=b_v s/n$. The excess measures how far the vertex deviates from the desired conflict size n/s. We check if $\sum_{v\in\mathrm{VD}(R\cup K)} t_v\log t_v\le T$. If not, we start over with a new sample. Below, we prove that for $T=\Theta(s)$, the success probability is at least $3/4$. The total success probability is $1/2$, and the expected number of attempts is 2. Thus, in expected time $O(n\log s+s\log s)$, we can find a sample $R\subseteq S$ with $\sum_{v\in\mathrm{VD}(R\cup K)} b_v=O(n)$ and $\sum_{v\in\mathrm{VD}(R\cup K)} t_v\log t_v=O(s)$.

We now analyze the success probabilities, using the classic Clarkson-Shor method. We begin with the following version of the Chazelle-Friedman bound [6].

Lemma 2 *Let X be a planar point set of size o, and let $Y\subset\mathbb{R}^2$ with $|Y|\le 3$. For fixed $p\in(0,1]$, let $R\subseteq X$ be a random subset of size po and let $R'\subseteq X$ be a random subset of size $p'o$, for $p'=p/2$. Suppose that $p'o\ge 4$. Fix $\mathbf{u}\in X^3$, and let $v_{\mathbf{u}}$ be the Voronoi vertex defined by $\mathbf{u}$. Let $b_{\mathbf{u}}$ be the number of points from X in the largest circle with center $v_{\mathbf{u}}$ and with no points from R in its interior. Then,*

$$\Pr[v_{\mathbf{u}}\in\mathrm{VD}(R\cup Y)]\le 64e^{-pb_{\mathbf{u}}/2}\Pr[v_{\mathbf{u}}\in\mathrm{VD}(R'\cup Y)].$$

Proof. Let $\sigma=\Pr[v_{\mathbf{u}}\in\mathrm{VD}(R\cup Y)]$ and $\sigma'=\Pr[v_{\mathbf{u}}\in\mathrm{DT}(R'\cup Y)]$. The vertex $v_{\mathbf{u}}$ is in $\mathrm{VD}(R\cup Y)$ precisely if $\mathbf{u}\subseteq R\cup Y$ and $B_{\mathbf{u}}\cap(R\cup Y)=\emptyset$, where $B_{\mathbf{u}}$ are the points from X in the conflict circle of $v_{\mathbf{u}}$. If $Y\cap B_{\mathbf{u}}\neq\emptyset$, then $\sigma=\sigma'=0$, and the lemma holds. Thus, assume that $Y\cap B_{\mathbf{u}}=\emptyset$. Let $d_{\mathbf{u}}=|\mathbf{u}\setminus Y|$, the number of points in $\mathbf{u}$ not in Y. There are $\binom{o-b_{\mathbf{u}}-d_{\mathbf{u}}}{po-d_{\mathbf{u}}}$ ways to choose a po-subset from X that avoids all points in $B_{\mathbf{u}}$ and contains all points of $\mathbf{u}\cap X$, so

$$\begin{aligned}\sigma&=\binom{o-b_{\mathbf{u}}-d_{\mathbf{u}}}{po-d_{\mathbf{u}}}\Big/\binom{o}{po}\\&=\frac{\prod_{j=0}^{po-d_{\mathbf{u}}-1}(o-b_{\mathbf{u}}-d_{\mathbf{u}}-j)}{\prod_{j=0}^{po-d_{\mathbf{u}}-1}(po-d_{\mathbf{u}}-j)}\Big/\frac{\prod_{j=0}^{po-1}(o-j)}{\prod_{j=0}^{po-1}(po-j)}\\&=\prod_{j=0}^{d_{\mathbf{u}}-1}\frac{po-j}{o-j}\cdot\prod_{j=0}^{po-d_{\mathbf{u}}-1}\frac{o-b_{\mathbf{u}}-d_{\mathbf{u}}-j}{o-d_{\mathbf{u}}-j}\\&\le p^{d_{\mathbf{u}}}\prod_{j=0}^{po-d_{\mathbf{u}}-1}\left(1-\frac{b_{\mathbf{u}}}{o-d_{\mathbf{u}}-j}\right).\end{aligned}$$

Similarly, we get

$$\sigma'=\prod_{i=0}^{d_{\mathbf{u}}-1}\frac{p'o-i}{o-i}\prod_{j=0}^{p'o-d_{\mathbf{u}}-1}\left(1-\frac{b_{\mathbf{u}}}{o-d_{\mathbf{u}}-j}\right),$$

and since $p'o\ge 4$ and $i\le 2$, it follows that

$$\sigma'\ge\left(\frac{p'}{2}\right)^{d_{\mathbf{u}}}\prod_{j=0}^{p'o-d_{\mathbf{u}}-1}\left(1-\frac{b_{\mathbf{u}}}{o-d_{\mathbf{u}}-j}\right).$$

Therefore, since $p'=p/2$,

$$\begin{aligned}\frac{\sigma}{\sigma'}&\le\left(\frac{2p}{p'}\right)^{d_{\mathbf{u}}}\prod_{j=p'o-d_{\mathbf{u}}}^{po-d_{\mathbf{u}}-1}\left(1-\frac{b_{\mathbf{u}}}{o-d_{\mathbf{u}}-j}\right)\\&\le 64\left(1-\frac{b_{\mathbf{u}}}{o}\right)^{po/2}\le 64\,e^{pb_{\mathbf{u}}/2}.\end{aligned}$$

□

We can now bound the total expected conflict size.

Lemma 3 *We have* $\mathbf{E}\left[\sum_{v\in \mathrm{VD}(R\cup K)} b_v\right] = O(n)$.

Proof. By expanding the expectation, we get

$$\mathbf{E}\left[\sum_{v\in \mathrm{VD}(R\cup K)} b_v\right] = \sum_{\mathbf{u}\in S^3} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R\cup K)] b_{\mathbf{u}},$$

$v_{\mathbf{u}}$ being the Voronoi vertex of $\mathbf{u}$ and $b_{\mathbf{u}}$ its conflict size. By Lemma 2 with $X = S$, $Y = K$ and $p = s/n$,

$$\leq \sum_{\mathbf{u}\in S^3} 64e^{-pb_{\mathbf{u}}/2} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)] b_{\mathbf{u}},$$

where $R' \subseteq S$ is a sample of size $s/2$. We estimate

$$\leq \sum_{t=0}^{\infty} \sum_{\substack{\mathbf{u}\in S^3 \\ b_{\mathbf{u}}\in[\frac{t}{p},\frac{t+1}{p})}} \frac{64e^{-t/2}(t+1)}{p} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)]$$

$$\leq \frac{1}{p} \sum_{\mathbf{u}\in S^3} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)] \sum_{t=0}^{\infty} 64e^{-t/2}(t+1)$$

$$= O(s/p) = O(n),$$

since $\sum_{\mathbf{u}\in S^3} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)] = O(s)$ is the size of $\mathrm{VD}(R'\cup K)$ and $\sum_{t=0}^{\infty} e^{-t/2}(t+1) = O(1)$. □

By Lemma 3 and Markov's inequality, there is an $M = \Theta(n)$ with $\Pr[\sum_{v\in \mathrm{VD}(R\cup K)} b_v > M] \leq 1/4$.

Lemma 4 $\mathbf{E}\left[\sum_{v\in \mathrm{VD}(R\cup K)} t_v \log t_v\right] = O(s)$.

Proof. By Lem. 2 with $X = S$, $Y = K$, and $p = s/n$,

$$\mathbf{E}\left[\sum_{v\in \mathrm{VD}(R\cup K)} t_v \log t_v\right]$$

$$= \sum_{\mathbf{u}\in S^3} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R\cup K)]\, t_{\mathbf{u}} \log t_{\mathbf{u}}$$

$$\leq \sum_{\mathbf{u}\in S^3} 64e^{-pb_{\mathbf{u}}/2} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)] t_{\mathbf{u}} \log t_{\mathbf{u}}$$

$$\leq \sum_{t=0}^{\infty} \sum_{\substack{\mathbf{u}\in S^3 \\ b_{\mathbf{u}}\in[\frac{t}{p},\frac{t+1}{p})}} 64e^{-\frac{t}{2}}(t+1)^2 \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)]$$

$$\leq \sum_{t=0}^{\infty} 64e^{-t/2}(t+1)^2 \sum_{\mathbf{u}\in S^3} \Pr[v_{\mathbf{u}} \in \mathrm{VD}(R'\cup K)]$$

$$= O(s).$$

□

By Markov's inequality and Lemma 4, there is a $T = \Theta(s)$ with $\Pr[\sum_{v\in \mathrm{VD}(R\cup K)} t_v \log t_v \geq T] \leq 1/4$. This finishes the first phase of the sampling.

Let $\alpha > 0$ be a sufficiently large constant. The next goal is to sample for each vertex v with $t_v \geq 2$ a random subset $R_v \subseteq B_v$ of size $\alpha t_v \log t_v$ (recall that B_v is the conflict set of v).

Lemma 5 *In total time $O(n \log s)$, we can sample for each vertex $v \in \mathrm{VD}(R\cup K)$ with $t_v \geq 2$ a random subset $R_v \subseteq B_v$ of size $\alpha t_v \log t_v$.*

Proof. First, we perform $O(s)$ rounds to sample for each vertex v with $t_v \geq 2$ a sequence I_v of $\alpha t_v \log t_v$ distinct numbers from $\{1, \ldots, b_v\}$. For this, we use the algorithm from Lemma 1 in parallel for each relevant vertex from $\mathrm{VD}(R\cup K)$. Since $\sum_v t_v \log t_v = O(s)$, this takes total time $O(s \log s)$ and total space $O(s)$.

After that, we scan through S. For each vertex v, we have a counter c_v, initialized to 0. For each $p \in S$, we find the conflict vertices of p, and for each conflict vertex v, we increment c_v. If c_v appears in the corresponding set I_v, we add p to R_v. The total running time is $O(n \log s)$, as we do one point location for each input point and the total conflict size is $O(n)$. □

We next show that for a *fixed* vertex $v \in \mathrm{VD}(R\cup K)$, with constant probability, all vertices in $\mathrm{VD}(R_v)$ have conflict size n/s with respect to B_v.

Lemma 6 *Let $v \in \mathrm{VD}(R\cup K)$ with $t_v \geq 2$, and let $R_v \subseteq B_v$ be the sample for v. The expected number of vertices v' in $\mathrm{VD}(R_v)$ with at least n/s points from B_v in their conflict circle is at most $1/2$.*

Proof. Recall that $t_v = b_v s/n$. We have

$$\mathbf{E}\left[\sum_{\substack{v'\in \mathrm{VD}(R_v) \\ b'_{v'}\geq n/s}} 1\right] = \sum_{\substack{\mathbf{u}\in B_v^3 \\ b'_{\mathbf{u}}\geq n/s}} \Pr[v'_{\mathbf{u}} \in \mathrm{VD}(R_v)],$$

where $b'_{\mathbf{u}}$ is the conflict size of $v'_{\mathbf{u}}$ with respect to B_v. Using Lemma 2 with $X = B_v$, $Y = \emptyset$, and $p = (\alpha t_v \log t_v)/b_v = \alpha(s/n)\log t_v$, this is

$$\leq \sum_{\substack{\mathbf{u}\in B_v^3 \\ b'_{\mathbf{u}}\geq n/s}} 64e^{-pb'_{\mathbf{u}}/2} \Pr[v'_{\mathbf{u}} \in \mathrm{VD}(R'_v)]$$

$$\leq 64e^{-(\alpha/2)\log t_v} \sum_{\mathbf{u}\in B_v^3} \Pr[v'_{\mathbf{u}} \in \mathrm{VD}(R'_v)]$$

$$= O(t_v^{-\alpha/2} t_v \log t_v) \leq 1/2,$$

for α large enough (remember that $t_v \geq 2$). □

By Lemma 6 and Markov's inequality, the probability that all vertices from $\mathrm{VD}(R_v)$ have at most $2n/s$ points from B_v in their conflict circles is at least $1/2$.

If so, we call v *good*. Scanning through S, we can identify the good vertices in time $O(n \log s)$ and space $O(s)$. The expected number of good vertices is $s'/2$, where s' is the size of $\mathrm{VD}(R \cup K)$. For the remaining vertices, we repeat the process with new random samples, but this time we take two samples per vertex, decreasing the failure probability to $1/4$. We repeat the process, taking in each round the maximum number of samples that fit into the work space. In general, if we have s'/a_i active vertices in round i, we can take a_i samples per vertex, resulting in a failure probability of 2^{-a_i}. Thus, the expected number of active vertices in round $i+1$ is $s'/a_{i+1} = s'/(a_i 2^{a_i})$. After $O(\log^* s)$ rounds, all vertices are good. To summarize:

Lemma 7 *In total expected time $O(n \log s \log^* s)$ and space $O(s)$, we can find sets $R \subseteq S$ and $R_v \subset B_v$ for each vertex $v \in \mathrm{VD}(R' \cup K)$ such that (i) $|R| = s$: (ii) $\sum_{v \in \mathrm{VD}(R \cup K)} |R_v| = O(s)$; and (iii) for every R_v, all vertices of $\mathrm{VD}(R_v)$ have at most $2n/s$ points from B_v in their conflict circle.*

We set $R_2 = R \cup \bigcup_{v \in \mathrm{VD}(R \cup K)} R_v$. By Lemma 7, $|R_2| = O(s)$. We compute $\mathrm{VD}(R_2 \cup K)$ and triangulate its bounded cells. For a triangle Δ of the triangulation, let $r \in R_2 \cup K$ be the site whose cell contains Δ, and v_1, v_2, v_3 the vertices of Δ. We set $B_\Delta = \{r\} \cup \bigcup_{i=1}^{3} B_{v_i}$. One can show that $|B_\Delta| = O(n/s)$.

Lemma 8 *For every triangle Δ in the triangulation of $\mathrm{VD}(R_2 \cup K)$, we have $\mathrm{VD}(S \cup K) \cap \Delta = \mathrm{VD}(B_\Delta) \cap \Delta$.*

Proof. Let v_1, v_2, v_3 be the vertices of Δ and let $r \in R_2 \cup K$ be the point whose cell contains Δ. Fix a point $x \in \Delta$, and let C be a circle with center x and no points from B_Δ in its interior. It suffices to show that C contains no points from $S \setminus B_\Delta$ in its interior. Suppose there exists a point $p \in S \setminus B_\Delta$ that lies inside of C. Consider the bisector B of p and r. Since C contains p but not r, we have $d(x,p) < d(x,r)$, so x lies on the same side of B as p. Since $x \in \Delta$, at least one of v_1, v_2, v_3, is on the same side of B as p; say v_1. This means that $d(v_1,p) < d(v_1,r)$, so p lies inside the circle around v_1 with r on the boundary. This is precisely the conflict circle of v_1. □

Theorem 9 *Let S be a planar n-point set. In expected time $O((n^2/s) \log s + n \log s \log^* s)$ and space $O(s)$, we can compute all Voronoi vertices of S.*

Proof. We compute a set R_2 as above. This takes $O(n \log s \log^* s)$ time and space $O(s)$. We triangulate the bounded cells of $\mathrm{VD}(R_2 \cup K)$ and compute a point location structure for the result. Since there are $O(s)$ triangles, we can store the resulting triangulation in the workspace. Now, the goal is to compute simultaneously for all triangles Δ the Voronoi diagram $\mathrm{VD}(B_\Delta)$ and to output all Voronoi vertices that lie in Δ and are defined by points from S. By Lemma 8, this gives all Voronoi vertices of $\mathrm{VD}(S)$.

Given a planar m-point set X, the algorithm by Asano et al. finds all vertices of $\mathrm{VD}(X)$ in $O(m)$ scans over the input, with constant workspace [2]. We can perform a simultaneous scan for all sets B_Δ by determining for each point in S all sets B_Δ that contain it. This takes total time $O(n \log s)$, since we need one point location for each $p \in S$ and since the total size of the B_Δ's is $O(n)$. We need $O(\max_\Delta |B_\Delta|) = O(n/s)$ such sweeps, so the second part of the algorithm needs $O((n^2/s) \log s)$ time. □

Acknowledgments. This work began while W. Mulzer, P. Seiferth, and Y. Stein visited the Tokuyama Laboratory at Tohoku University. We would like to thank Takeshi Tokuyama and all members of the lab for their hospitality and for creating a conducive and stimulating research environment. W. Mulzer is partially supported by DFG grants MU 3501/1 and MU 3501/2. P. Seiferth is partially supported by DFG Grant MU 3501/1. Y. Stein is supported by the DFG within the research training group 'Methods for Discrete Structures' (GRK 1408).

References

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

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

[3] L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, pages 1–33, 2014.

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

[5] A. Borodin and S. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11:287–297, 1982.

[6] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[7] J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theoret. Comput. Sci.*, 12:315–323, 1980.

[8] J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theoret. Comput. Sci.*, 165(2):311–323, 1996.

[9] J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 264–268, 1998.

[10] I. Pohl. A minimum storage algorithm for computing the median. Technical Report RC2701, IBM, 1969.

[11] J. E. Savage. *Models of computation—exploring the power of computing.* Addison-Wesley, 1998.

Linear-Time Algorithms for the Farthest-Segment Voronoi Diagram and Related Tree Structures*

Elena Khramtcova† Evanthia Papadopoulou†

Abstract

We present linear-time algorithms to construct Voronoi diagrams with disconnected regions, whose graph structure is a tree, after the sequence of their faces along an enclosing boundary (or at infinity) is known. We illustrate our approach on the farthest-segment Voronoi diagram; however, it is also applicable to computing the order-$(k+1)$ subdivision within an order-k segment Voronoi region, and to updating the nearest-neighbor Voronoi diagram after deletion. It is also applicable to other sites and metrics.

1 Introduction

It is well known that the Voronoi diagram of points in convex position can be computed in linear time, given their convex hull [1]. The same holds for a class of related diagrams such as the farthest-point Voronoi diagram, computing the medial axis of a convex polygon, and deleting a point from the nearest-neighbor Voronoi diagram. In an abstract setting, a *Hamiltonian abstract Voronoi diagram* [8] can be computed in linear time, given the order of Voronoi regions along an unbounded simple curve, which visits each region exactly once, and which can intersect each bisector only once. This linear construction has recently been extended to include forest structures [4] under the same conditions of no repetition. The medial axis of a simple polygon can also be computed in linear time [6]. It is therefore natural to ask what other types of Voronoi diagrams can be constructed in linear time. In this paper we address tree-like Voronoi diagrams with disconnected regions.

Classical variants of Voronoi diagrams, such as higher-order Voronoi diagrams, for sites other than points, had been surprisingly ignored in the computational geometry, until recently [3, 12]. Given a set S of n simple geometric objects in the plane, called sites, the *order-k Voronoi diagram* of S is a partitioning of the plane into regions, such that every point within a region has the same k nearest sites. For $k = 1$, this is the *nearest-neighbor Voronoi diagram*; for $k = n-1$ it is *farthest-site Voronoi diagram* of S. Despite similarities, these diagrams illustrate fundamental structural differences from their counterparts for points, such as the presence of disconnected regions (see also [2, 9]).

In this abstract we give linear-time algorithms (expected and deterministic) for constructing the farthest-segment Voronoi diagram, after the sequence of its faces at infinity is known. The method applies to constructing the order-$(k+1)$ subdivision within an order-k Voronoi region, and updating the nearest-neighbor Voronoi diagram after deletion. Interestingly, the latter problem requires computing initially two different tree-like diagrams. A major difference from the respective problems for points is that the sequence of faces along a relevant enclosing boundary forms a Davenport-Schinzel sequence of order at least 2,[1] unlike points, where no repetition can exist.

The segment counterpart of higher-order Voronoi diagrams is an important variant for a variety of applications dealing with polygonal objects or embedded planar graphs, see e.g., [11] and references therein.

2 Preliminaries and Definitions

Let S be a set of arbitrary line segments in $\mathbb{R}^2$, that may intersect or touch at a single point. The distance between a point q and a line segment s_i is $d(q, s_i) = \min\{d(q, y), \forall y \in s_i\}$, where $d(q, y)$ denotes the ordinary distance between two points q, y. The bisector of two segments $s_i, s_j \in S$ is $b(s_i, s_j) = \{x \in \mathbb{R}^2 \mid d(x, s_i) = d(x, s_j)\}$. For disjoint segments, it is an unbounded curve that consists of a constant number of pieces, where each piece is portion of an elementary bisector between the endpoints and open portions of s_i, s_j. If two segments intersect at point p, their bisector consists of two such curves intersecting at p.

The farthest Voronoi region of a segment s_i is $\mathit{freg}(s_i) = \{x \in \mathbb{R}^2 \mid d(x, s_i) > d(x, s_j), 1 \leq j \leq n, j \neq i\}$. The regions of all the segments in S together with their bounding edges and vertices, define a partition of the plane, called the *farthest-segment Voronoi diagram FVD(S)*, see Fig. 1(a). Any maximally-connected subset of a Voronoi region is called a *face*.

A farthest Voronoi region $\mathit{freg}(s_i)$ is non-empty and unbounded in direction ϕ if and only if there exists

*Research supported in part by the Swiss National Science Foundation, project 20GG21-134355, under the ESF EUROCORES program EuroGIGA/VORONOI.

†Faculty of Informatics, Università della Svizzera italiana (USI), Lugano, Switzerland.

[1] Order-3 for the farthest-segment Voronoi diagram, order-2 for disjoint segments or the farthest abstract Voronoi diagram, and order-4 for the order-k Voronoi diagram of segments.

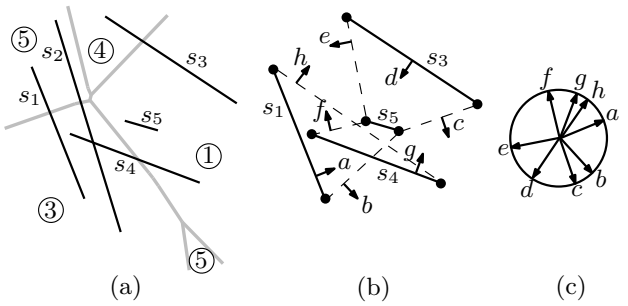


Figure 1: [10] (a) FVD(S), $S = \{s_1, \ldots, s_5\}$; (b) its farthest hull; (c) Gmap(S)

an open halfplane, normal to ϕ, which intersects all segments in S but s_i [2]. The line ℓ, normal to ϕ, bordering such a halfplane is called a *supporting line*. The direction ϕ (normal to ℓ) is denoted as $\nu(\ell)$ and it is referred to as the *hull direction* of ℓ. An unbounded Voronoi edge between $freg(s_i)$ and $freg(s_j)$ is a portion of $b(p, q)$, where p, q are endpoints of s_i and s_j, such that the line through $\overline{pq}$ induces an open halfplane that intersects all segments in S, except s_i, s_j (and possibly except additional segments incident to p, q). Segment $\overline{pq}$ is called a *supporting segment*; the direction normal to $\overline{pq}$ pointing to the inside of this halfplane is denoted as $\nu(\overline{pq})$ and it is referred to as the *hull direction* of $\overline{pq}$. A segment $s_i \in S$ such that the line ℓ through s_i is supporting, is called a *hull segment*; its hull direction is $\nu(s_i) = \nu(\ell)$, Fig. 1(a) and (b) illustrate a farthest-segment Voronoi diagram and its hull respectively. In Fig. 1(b), dashed lines show the supporting segments, and the hull segments are shown in bold. Arrows show the hull directions of all supporting and hull segments.

The Gaussian map of FVD(S) (Gmap(S)) provides a correspondence between the faces of FVD(S) and a circle of directions K (see Fig. 1(c)) [10]. Each Voronoi face is mapped to an arc on K, which represents a set of directions along which the face is unbounded. The Gmap(S) can be viewed as a cyclic sequence of arcs, where each arc corresponds to one face of FVD(S). Two neighboring arcs are separated by the hull direction of a supporting segment, which corresponds to an unbounded Voronoi edge. The arc of a hull segment consists of two sub-arcs separated by the hull direction ν of the segment. Gmap(S) can be computed in $O(n \log n)$ time (or output-sensitive $O(n \log h)$, where $h = |\text{Gmap}(S)|$) [10].

The standard point-line duality transformation t offers a correspondence between the faces of *FVD(S)* and envelopes of *wedges*. A segment $s_i = uv$ is sent into a *lower* wedge below (see e.g., Fig. 2) and an *upper* wedge above $t(u)$ and $t(v)$ respectively. Let E (resp., E') be the boundary of the union of the lower (resp., upper) wedges. There is a clear correspondence between E (resp., E') and the *upper* (resp., *lower*) Gmap: the vertices of E are exactly the hull directions of supporting segments on the upper Gmap and the apexes of wedges in E are exactly the hull directions of hull segments. In fact, any x-monotone path p in the arrangement of upper (resp., lower) wedges can be transformed into a sequence of arcs in the upper (resp., lower) circle of directions, i.e., the portion of K above (resp., below) its horizontal diameter.

3 The Farthest Voronoi Diagram of a Sequence

Let G be a sequence of arcs on the circle of directions K corresponding to a pair of x-monotone paths, one in the arrangement of upper wedges and one in the arrangement of lower wedges in dual space. G is called an arc sequence. For an arc α in G, let $s_\alpha \in S$ be the segment associated with it.

Given a point x, $x \notin s_\alpha$, consider the ray emanating from s_α that realizes the Euclidean distance between s_α and x. Let $r(x, s_\alpha)$ be the unbounded portion of this ray, starting at x and extending to infinity away from s_α. We say that x is *attainable* from α if the direction of $r(x, s_\alpha)$ is contained in α. The point(s) $x \in s_\alpha$ that induce α are always attainable from α. Let $d(x, \alpha) = d(x, s_\alpha)$ if x is attainable from α; and let $d(x, \alpha) = -\infty$ otherwise. The locus of points attainable from α is referred to as the *attainable region* of α, $R(\alpha)$. The bisector between two arcs α, γ ($s_\alpha \neq s_\gamma$) is $b(\alpha, \gamma) = b(s_\alpha, s_\gamma) \cap R(\alpha) \cap R(\gamma)$. The farthest Voronoi region of α can now be defined in the ordinary way as $freg(\alpha) = \{x \in \mathbb{R}^2 \mid d(x, \alpha) > d(x, \gamma), \forall \text{ arc } \gamma \in G, \gamma \neq \alpha\}$.

The farthest Voronoi diagram of G, FVD(G), is the subdivision of the plane obtained by these regions and their boundaries. Let $T(G) = \mathbb{R}^2 \setminus \cup_{\alpha \in G} freg(\alpha)$. If all edges of $T(G)$ are portions of arc bisectors then G, $T(G)$, and FVD(G) are all called *proper*. For an arbitrary G, $T(G)$ may contain boundaries of attainable regions and FVD(G) may even contain holes. The diagrams computed by our algorithms are all proper.

Lemma 1 *For a proper arc sequence G, $T(G)$ is a (connected) tree.*

G is called a *subsequence* of Gmap(S) if every arc in G entirely contains a corresponding arc in Gmap(S) induced by the same segment. The arcs in G are expanded versions of the arcs in Gmap(S), and they are called *original arcs*. A sequence G' is called an *augmented subsequence* of Gmap(S) if G' contains at least one original arc for every segment that appears in G'. Hence, G' consists of *original* and *new* arcs. If G' contains the same original arcs as G then G' is said to be *corresponding* to G. Note that G and G' are no longer envelopes of wedges, but simply x-monotone paths, where (assuming lower wedges) the path of G' is above (or on) the path of G.

A subsequence G is derived from Gmap(S) by deletion of arcs. When deleting an arc β, the neighboring arcs α and γ *expand* over β. Either both α and γ expand (see Fig. 2(a) for segments in the dual space, or Fig. 3(a)-(c)) or one expands while the other shrinks (see Fig. 2(b)). By the definition of FVD(G), both α, γ remain in $G \setminus \{\beta\}$ and their common endpoint is the hull direction of their common supporting segment, $\nu(\alpha, \gamma)$. If $s_\alpha = s_\gamma$, let $\nu(\alpha, \gamma) = \nu(s_\beta)$. Direction $\nu(s_\beta)$ corresponds to an *artificial* bisector $b(\alpha, \gamma)$ as defined below in Def. 1. If $s_\alpha = s_\beta$ then α expands to cover the entire β and $\nu(\alpha, \gamma) = \nu(\beta, \gamma)$.

Consider now the result of inserting back β between (the expanded) arcs α, γ (see Fig. 2 illustrating the corresponding segments in dual space). The result is $\alpha\beta\gamma$, if both α, γ expanded when removing β, or $\alpha\gamma'\beta\gamma$, if α shrunk, or $\alpha\beta\alpha'\gamma$, if γ shrunk, where α' and γ' are new arcs of s_α and s_γ respectively. Thus, re-inserting β in subsequence $G \setminus \{\beta\}$ results in an augmented subsequence $G' = (G \setminus \{\beta\}) \cup \{\beta\}$, which may be different from G.

Definition 1 *Let α, β, γ be consecutive arcs in G', where $s_\alpha = s_\gamma$. Let the artificial bisector $b(\alpha, \gamma)$ (for G') be a ray in the direction of $\nu(s_\beta)$, emanating from the relevant endpoint p_α of s_α (see Fig. 3(d)).*

4 A Randomized Linear Construction

We give an expected linear-time algorithm to compute FVD(S), given Gmap(S). It is inspired by the two-phase randomized approach of [5] for points in convex position, however, it is augmented considerably to handle non-uniqueness and new elements in the cyclic sequence of arcs.

Let $\alpha_1, \alpha_2, \ldots, \alpha_h$ be a random permutation of arcs in Gmap(S), and let $A_i = \{\alpha_1, \alpha_2, \ldots, \alpha_i\}$, $1 \le i \le h$, be the set of the first i arcs in this order. The algorithm proceeds in two phases. Let t be the largest index such that $\{\alpha_1, \ldots, \alpha_t\}$ consists of arcs of two segments resulting in exactly two *maximal* arcs, where consecutive arcs of the same segment are joined into one maximal arc. Phase 1 computes the subsequence G_i, $t \le i < h$, where $G_h = \text{Gmap}(A_h)$, and G_i is obtained from G_{i+1} by removing arc α_{i+1} as described in Sec. 3. The two neighbors of α_{i+1} in G_{i+1} are recorded as a tentative re-entry point for α_{i+1} in phase 2. In phase 2, the algorithm computes incrementally G'_i and FVD(G'_i), for $t < i \le h$, starting with FVD(G'_t), $G'_t = G_t$. G'_{i+1} is the sequence obtained from G'_i by inserting back arc α_{i+1}.During the re-entry of α_{i+1} a new arc may be created. As a result, $G'_i \neq G_i$; however, G'_i is an augmented subsequence of Gmap(S). Because of new arcs, the two recorded neighbors of α_{i+1} from phase 1 need not be neighbors of α_{i+1} in G'_{i+1}. Thus, scanning a number of new arcs may be required to identify an entry point for α_{i+1}.

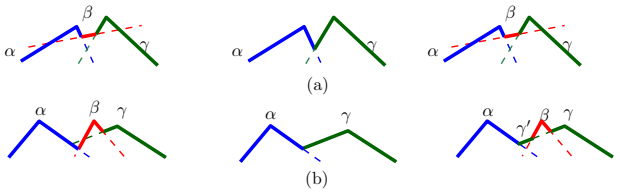


Figure 2: Deleting and re-inserting β in sequence $\alpha\beta\gamma$. (a) both α and γ enlarge; (b) γ enlarges and α shrinks. From left to right: the initial sequence $\alpha\beta\gamma$; the result of removing β; the result of re-inserting β.

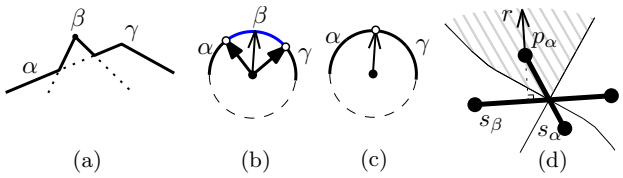


Figure 3: Sequence $\alpha\beta\gamma$ when s_α, s_γ (a) The dual wedges; (b) G_{i+1}; (c) G_i; (d) The ray r splitting *freg*(α) and *freg*(γ).

The entry point is either an unbounded bisector (regular or artificial), which is deleted from FVD(G'_{i+1}), or an arc σ, which entirely contains α_{i+1}. In the latter case, *freg*(σ) is split into two faces and *freg*(α_{i+1}) is inserted within; a new arc σ' is created as a result of the split. At the end of phase 2, we obtain FVD(G'_h) = FVD(S) ($G'_h = G_h$).

Lemma 2 *The number of arcs in G'_i is at most $2i$. Thus, the complexity of* FVD(G'_i) *is $O(i)$.*

Lemma 3 *The expected number of new arcs traced at any step of phase 2 is constant (at most 1).*

Using backwards analysis we can prove that FVD(S) can be computed in $O(h)$ expected time, given Gmap(S).

5 A Deterministic Linear Divide-and-Conquer

We now augment the framework of Aggarwal et al. [1] for points in convex position with techniques from Secs. 3, 4, and derive a linear-time algorithm to compute the farthest Voronoi diagram, given Gmap(S). Let G be a subsequence of Gmap(S), and let G' be a corresponding augmented subsequence such that the complexity of G' is bounded by $O(|G|)$. The flow of the algorithm follows [8], which in turn follows [1].

1. Unite any consecutive arcs in G of the same segment into a single maximal arc for that segment.
2. Color each element of G as *red* or *blue*, by applying the following two rules:

   (a) For each 5-tuple F of consecutive arcs $\{\alpha, \beta, \gamma, \delta, \epsilon\}$ in G, compute FVD(F') by the algorithm

of Sec. 4 in the fixed deletion order $\alpha, \epsilon, \beta, \delta, \gamma$. Color γ as red, if *freg*(γ) did not change after the insertion of ϵ and α; otherwise, color γ as blue.

(b) For each series of consecutive blue arcs, color red every other arc except the last arc.

3. Let B be the blue sequence as obtained from G by deleting all red arcs. Recursively compute FVD(B'). (B' is an augmented version of B.)
4. Re-color as *crimson* at least a constant fraction of the red arcs, such that for any two crimson arcs, if they were inserted in FVD(B'), their Voronoi regions would not be neighboring.
5. Insert the crimson arcs one by one in FVD(B'), resulting in FVD(V').
6. Let Gr (garnet) be the sequence obtained from G by deleting all blue and crimson arcs. Recursively compute FVD(Gr').
7. Merge FVD(V') and FVD(Gr') into FVD(G').
8. For any arcs that were united in Step 1, subdivide their regions in FVD(G') into finer parts by inserting the corresponding artificial bisectors.

The recursion ends when G has at most two maximal arcs. The appearance of new arcs requires a new handling for several steps of the algorithm. The number of new arcs must always remain bounded.

At the end of Step 2, the following holds: (1) No two consecutive arcs in G are red and no three consecutive arcs in G are blue. (2) For any two consecutive red arcs α, β in G, if α and β were inserted in FVD(B'), their Voronoi regions would be disjoint. These statements can be proven following the spirit of [8, Lemmas 8 and 9], however, the appearance of new arcs and the dependence of the result on the deletion order requires a new handling. The order of arc deletion given in Step 2 is such as to keep bounded the number of new arcs that surround γ in F'.

Step 4 can be performed in $O(|G'|)$ time by applying the *combinatorial lemma* of [1] to $T(B')$. Any red arc β is associated with a unique leaf of $T(B')$, which serves as an entry point for re-inserting β in FVD(B'). The insertion of β may split an arc of B' into two, in which case the entry point for β is an artificial bisector.

Step 7 can be performed in time $O(|G|)$, by keeping the generation of new arcs bounded, i.e., by keeping $|G'| = O(|G|)$. To this goal, we use a specific merging scheme, which ignores some of the merge curves. Our merging scheme guarantees that (a) G' contains all the original arcs of V' and Gr'; and (b) the number of new arcs created by merging is at most the number of original arcs in V' and Gr', which is $O(|G|)$. In addition, the insertion of a crimson arc in Step 5 causes at most one new arc. Thus, $|G'| = O(|G|)$.

By the procedure of Steps 2 and 4, there exist constants $q_1, q_2 > 0$ such that $|B| \leq q_1|G|$, $|Gr| \leq q_2|G|$, and $q_1 + q_2 < 1$. The time complexity is $T(m) \leq T(q_1 m) + T(q_2 m) + O(m)$, where $m = |G|$. This implies $T(m) = O(m)$, see [1].

Theorem 4 *Given* Gmap(S), *the* FVD(S) *can be computed in additional* $O(h)$ *time, where* h *is the number of faces in FVD(S).*

Theorem 4 applies also to updating a nearest-neighbor segment Voronoi diagram after the deletion of one site, and to computing the order-(k+1) subdivision within an order-k segment Voronoi region in time proportional to the complexity of the related region. We conjecture that it also applies to the respective abstract Voronoi diagrams, which we are currently investigating with several results so far to the affirmative. Note that the farthest abstract Voronoi diagram can be constructed in expected $O(n \log n)$ time by a randomized incremental construction [9], which is not related to the randomized approach in this abstract.

References

[1] Aggarwal, A., Guibas, L., Saxe, J., Shor, P.: A linear-time algorithm for computing the Voronoi diagram of a convex polygon. Discrete Comput. Geom. 4, 591–604 (1989)

[2] Aurenhammer, F., Drysdale, R., Krasser, H.: Farthest line segment Voronoi diagrams. Inform. Process. Lett. 100, 220–225 (2006)

[3] Bohler, C., Cheilaris, P., Klein, R., Liu, C.H., Papadopoulou, E., Zavershynskyi, M.: On the complexity of higher order abstract Voronoi diagrams. In: Proc. 40th ICALP. LNCS 7965, 208–219 (2013)

[4] Bohler, C., Klein, R., Liu, C.H.: Forest-like abstract Voronoi diagrams in linear time. In: CCCG (2014)

[5] Chew, L.P.: Building Voronoi diagrams for convex polygons in linear expected time. Tech. rep., Dartmouth College, Hanover, USA (1990)

[6] Chin, F., Snoeyink, J., Wang, C. A.: Finding the medial axis of a simple polygon in linear time. Discrete Comput. Geom., 21(3), 405–420 (1999)

[7] de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer-Verlag, 3rd edn. (2008)

[8] Klein, R., Lingas, A.: Hamiltonian abstract Voronoi diagrams in linear time. In: Proc. 9th ISAAC. LNCS 834, 11–19 (1994)

[9] Mehlhorn, K., Meiser, S., Rasch, R.: Furthest site abstract Voronoi diagrams. Int. J. Comput. Geom. Ap. 11(6), 583–616 (2001)

[10] Papadopoulou, E., Dey, S.: On the farthest line-segment Voronoi diagram. Int. J. Comput. Geom. Ap. 23(6), 443–460 (2013)

[11] Papadopoulou, E.: Net-aware critical area extraction for opens in VLSI circuits via higher-order Voronoi diagrams. IEEE T. CAD 30(5), 704–717 (2011)

[12] Papadopoulou, E., Zavershynskyi, M.: The higher-order Voronoi diagram of line segments. Algorithmica DOI 10.1007/s00453-014-9950-0 (2014)

β-skeletons for a set of line segments in R^{2*}

Abstract

β-skeletons are well-known neighborhood graphs for a set of points. We extend this notion to sets of line segments and present algorithms computing such skeletons for the entire range of β values. The main reason for such an extension is a study of β-skeletons for points moving along given line segments. We show that relations between 1-skeleton (Gabriel Graph), 2-skeleton (Relative Neighborhood Graph) and the Delaunay triangulation for sets of points hold also for sets of segments. We present algorithms for computing circle and lune-based β-skeletons. We describe an algorithm that for $\beta \geq 1$ computes the β-skeleton for a set S of n segments in the Euclidean plane in $O(n^2\alpha(n)\log n)$ time in the circle-based case and in $O(n^2\lambda_4(n))$ in the lune-based one, where the construction relies on the Delaunay triangulation for S. When $0 < \beta < 1$, the β-skeleton can be constructed in a $O(n^3\lambda_4(n))$ time.

1 Introduction

Our definition of the β-skeleton for line segments is based on the following definition of the β-skeletons for sets of points in the Euclidean space [9]:

Definition 1 *[9] For a given set of points $V = \{v_1, v_2, \ldots, v_n\}$ in $\mathbb{R}^2$, a distance function d and parameter $0 < \beta < \infty$ we define graph $G_\beta(V)$ – called a lune-based β-skeleton – as follows: two points $v', v'' \in V$ are connected with an edge if and only if no point from $V \setminus \{v', v''\}$ belongs to the set $N(v', v'', \beta)$ (neighborhood) where:*

1. *for $0 < \beta < 1$, $N(v', v'', \beta)$ is the intersection of two discs, each of them has radius $\frac{d(v',v'')}{2\beta}$ and whose boundaries contain both v' and v'';*
2. *for $1 \leq \beta < \infty$, $N(v', v'', \beta)$ is the intersection of two discs, each with radius $\frac{\beta d(v',v'')}{2}$, whose centers are in points $(\frac{\beta}{2})v' + (1 - \frac{\beta}{2})v''$ and in $(1 - \frac{\beta}{2})v' + (\frac{\beta}{2})v''$, respectively.*

The region $N(v', v'', \beta)$ is called a lune and points $v', v'' \in V$ are its generators.

*This research is supported by the ESF EUROCORES program EUROGIGA, CRP VORONOI.

Another form of β-neighborhoods has been studied for $\beta \geq 1$ (see for example [9]) leading to a different family of β-skeletons called circle-based β-skeletons. In this case, set $N^c(v', v'', \beta)$ is an union of two discs, each with radius $\frac{d(v',v'')}{2\beta}$ and having the segment $v'v''$ as a chord.

Hurtado, Liotta and Meijer [8] presented an $O(n^2)$ algorithm for the β-skeleton when $\beta < 1$. The lune-based β-skeletons for $1 \leq \beta \leq 2$ can be found in $O(n \log n)$ time [12]. For $\beta > 1$, the circle-based β-skeletons can be constructed in $O(n \log n)$ time too [5]. But so far the fastest algorithm for computing the lune-based β-skeletons for $\beta > 2$ runs in $O(n^{\frac{3}{2}} \log^{\frac{1}{2}} n)$ time [10].

Geometric structures concerning a set of line segments, e.g. the Voronoi diagram [3, 11] or the straight skeleton [1] are well-studied in the literature. Chew and Kedem [4] defined the Delaunay triangulation for line segments. Their definition was generalized by Brévilliers et al. [2]. However, β-skeletons for a set of line segments were completely unexplored. This paper makes an initial effort to fill this gap.

Let us consider the case when we compute the β-skeleton for a set of n points V where every point $v \in V$ is allowed to move along a straight-line segment s_v. Let $S = \{s_v | v \in V\}$. For each pair of segments s_{v_1}, s_{v_2} containing points $v_1, v_2 \in V$, respectively, we want to find such positions of points v_1 and v_2 that $s_v \cap N(v_1, v_2, \beta) = \emptyset$ for any $s_v \in S \setminus \{s_1, s_2\}$. We will attempt to solve this problem by defining a β-skeleton for the set of line segments S.

Let S be a finite set of disjoint closed line segments in a two-dimensional plane $\mathbb{R}^2$ with the Euclidean metric and distance function d. We extend Definition 1 to define the β-skeleton for the set S.

Definition 2 *$G_\beta(S)$ is a graph with n vertices corresponding to the segments in S, and edges connecting $s', s'' \in S$ if and only if there exist points $v' \in s'$ and $v'' \in s''$ such that $s \cap N(v', v'', \beta) = \emptyset$ for any $s \in S \setminus \{s', s''\}$.*

Note that when segments degenerate to points, we get the standard β-skeletons for point sets.

2 Preliminaries

Now, let the Delaunay triangulation $DT(S)$ for the set of line segments S be the graph (multigraph) dual to the Voronoi diagram for S (see [2]). We assume that the segments in S are in general position, i.e. no three segment endpoints are collinear and no circle is tangent to four segments.

Kirkpatrick and Radke [9] proved an important theorem connecting β-skeletons for a set of points V with the Delaunay triangulation $DT(V)$ ($RNG(V) \subseteq GG(V) \subseteq DT(V)$).

Definitions of the β-skeleton and the Delaunay triangulation for a set of line segments S preserve the inclusions from this theorem. We define $RNG(S)$ ($GG(S)$, respectively) as a lune-based 2-skeleton (1-skeleton, respectively).

Theorem 1 *Let us assume that line segments in S are in general position and let $G_\beta(S)$ ($G^c_\beta(S)$, respectively) denote the lune-based (circle-based, respectively) β-skeleton for a set S. For $1 \leq \beta < \beta'$ following inclusions hold true: $G_{\beta'}(S) \subseteq G_\beta(S) \subseteq GG(S) \subseteq DT(S)$ ($G^c_{\beta'}(S) \subseteq G^c_\beta(S) \subseteq GG(S) \subseteq DT(S)$, respectively).*

Proof. First we prove that $GG(S) \subseteq DT(S)$. Let $v_1 \in s_1, v_2 \in s_2$ be such a pair of points that there exists a disc D with diameter v_1v_2 containing no points belonging to segments from $S \setminus \{s_1, s_2\}$ inside of it. We transform D by homothety with respect to v_1 so that its image D' could be tangent to s_2 in the point t. Then we transform D' by homothety with respect to t so that its image D'' is tangent to s_1 (see Figure 1). The disc D'' lies inside of D, i.e. it does not intersect segments from $S \setminus \{s_1, s_2\}$, and is tangent to s_1 and s_2. Hence, if the edge s_1s_2 belongs to $GG(S)$ then it also belongs to $DT(S)$.

Let c_1, c_2 be centers of discs determining a lune $N \in N(s_1, s_2, \beta)$. Let c'_1 (c'_2, respectively) be an image of c_1 (c_2, respectively) by homothety with the factor $\frac{\beta'}{\beta}$ with respect to point v_1 (v_2, respectively). Then c'_1, c'_2 are centers of discs determining a lune $N' \in N(s_1, s_2, \beta')$ and $N \subseteq N'$. Hence $G_{\beta'}(S) \subseteq G_\beta(S)$.

The sequence of inclusions for circle-based β-skeletons is a straightforward consequence of fact that two different circles have at most two intersection points.

□

3 Algorithm computing β-skeletons for $0 < \beta < 1$

Let S be a set of n disjoint line segments in the Euclidean plane.

Observation 1 *For a given parameter $0 < \beta < 1$ if v is a point from the boundary of a lune $N(v_1, v_2, \beta)$, different than v_1 and v_2, then an angle $\angle v_1vv_2$ has a constant measure which depends only on β.*

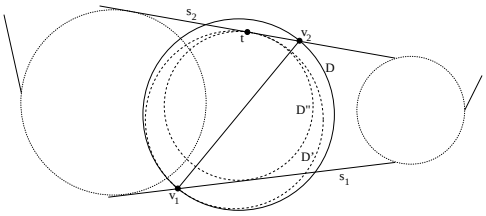


Figure 1: $GG(S) \subseteq DT(S)$.

Let us consider a set of parametrized lines containing given segments. A line $P(s_i)$ contains a segment $s_i \in S$ and has parametrization $q_i(t_i) = (x^i_1, y^i_1) + t_i[x^i_2 - x^i_1, y^i_2 - y^i_1]$, where (x^i_1, y^i_1) and (x^i_2, y^i_2) are ends of the segment s_i and $t_i \in \mathbb{R}$.

Let s_1 and s_2 be generators of a lune and δ be the measure of an inscribed angle defining a lune for a given value of β. The main idea of the algorithm is as follows. For any point $v_1 \in P(s_1)$ we compute points $v_2 \in P(s_2)$ for which there exists a point $v \in P(s)$, where $s \in S \setminus \{s_1, s_2\}$, such that $\delta \leq \angle v_1vv_2 \leq 2\pi - \delta$, i.e. $v \in N(v_1, v_2, \beta)$ (see Figure 2). Then we analyze an union of results for all $s \in S \setminus \{s_1, s_2\}$. If it contains all pairs of points (v_1, v_2), where $v_1 \in s_1$ and $v_2 \in s_2$, then $(s_1, s_2) \notin G_\beta(S)$.

For a given $t_1 \in \mathbb{R}$ we shoot rays from a point $v_1 = q_1(t_1) \in P(s_1)$ towards $P(s)$. Let us assume that a given ray intersects some segment $s \in S \setminus \{s_1, s_2\}$ in a point $v = q(t)$ for some value of $t \in \mathbb{R}$. Let $w(t) = \vec{v_1v}$ be the vector between points v_1 and v. Then $w(t) = [A_1t + B_1t_1 + C_1, A_2t + B_2t_1 + C_2]$ where coefficients A_i, B_i, C_i for $i = 1, 2$ depend only on endpoints coordinates of segments s_1 and s. The ray refracts in v from the segment s in a such way that the angle between directions of incidence and refraction of the ray is equal to the angle δ. The parametrized equation of the refracted ray is $r(z, t) = v + z \cdot R_\delta w(t)$ for $z \geq 0$ (or $r(z, t) = v + z \cdot R'_\delta w(t)$ for $z \geq 0$, respectively) where R_δ (R'_δ, respectively) denotes a rotation matrix for a clockwise (counter-clockwise, respectively) angle δ. If refracted ray $r(z, t)$ intersects line $P(s_2)$ in a point $q_2(t_2) = r(z, t)$ (it is not always possible - see Figure 2) then we get $t_2(t) = \frac{M \cdot t^2 + p_1(t_1) \cdot t + p_2(t_1)}{N \cdot t + p_3(t_1)}$, where p_1, p_2 and p_3 are (at most quadratic) polynomials of variable t_1 and M, N are fixed.

Note that if we consider clockwise and counter-clockwise refraction separately then for a given point $q_1(t_1)$ and a given segment s, the graph of function t_2 with respect to t consists of parts of a hyperbola. We analyze a graph of correlations between variables t and t_2 (i.e. a set of pairs $(t, t_2(t))$ for fixed t_1) for both kinds of refractions - see Figure 2.

Let $T(t_1, s, s_2)$ be a set of all t_2 such that, for given t_1, points $q_1(t_1)$ and $q_2(t_2)$ generate a lune inter-

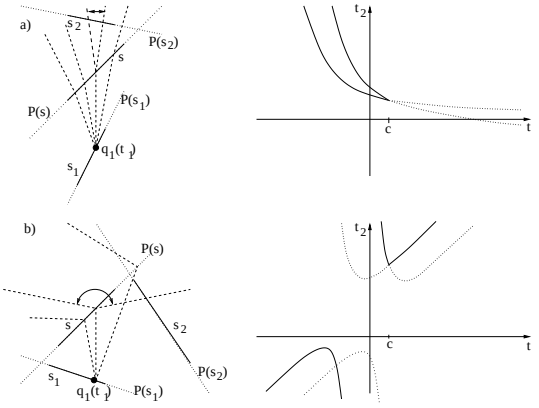


Figure 2: Examples of correlation between parameters t and t_2 (for fixed t_1) for (a) a refraction angle near to π (the arc with arrows shows, where for given ray direction are located points, which define a lune containing the refraction point) and (b) near to $\frac{\pi}{2}$. The value c corresponds to the intersection point of lines $P(s)$ and $P(s_2)$. Dotted line shows a situation when a line containing a refracted ray intersects $P(s_2)$ but the ray itself does not.

sected by segment s. Let $F(s_1, s, s_2) = \bigcup_{t_1 \in R} \{t_1\} \times T(t_1, s, s_2)$ be a set of pairs of parameters (t_1, t_2) such that some segment s intersects a lune generated by segments s_1 and s_2. The set $F(s_1, s, s_2)$ is an area limited by $O(1)$ algebraic curves of degree at most 3 (i.e. hyperbolas for $t = 0$ and $t = 1$ and envelopes of a set of hyperbolas for $0 < t < 1$, see Figure 3).

Lemma 2 *The edge s_1, s_2 belongs to the β-skeleton $G_\beta(S)$ if and only if*
$[0,1] \times [0,1] \setminus \bigcup_{s \in S \setminus \{s_1, s_2\}} F(s_1, s, s_2) \neq \emptyset$.

Proof. If $[0,1] \times [0,1] \setminus \bigcup_{s \in S \setminus \{s_1, s_2\}} F(s_1, s, s_2) \neq \emptyset$ then there exists a pair of parameters $(t_1, t_2) \in [0,1] \times [0,1]$ such that a lune generated by points $q_1(t_1) \in s_1$ and $q_2(t_2) \in s_2$ is not intersected by any segment $s \in S \setminus \{s_1, s_2\}$, i.e. $(s_1, s_2) \in G_\beta(S)$. The opposite implication can be proved in the same way. □

Theorem 3 *For $0 < \beta < 1$ the β-skeleton $G_\beta(S)$ can be found in $O(n^3 \lambda_4(n))$ time, where $\lambda_4(n)$ denotes the maximum possible length of a $(n, 4)$ Davenport-Schinzel sequence.*

Proof. We analyze $O(n^2)$ pairs of line segments. For each pair of segments s_1, s_2 we compute $\bigcup_{s \in S \setminus \{s_1, s_2\}} F(s_1, s, s_2)$. For each $s \in S \setminus \{s_1, s_2\}$ we find a set of pairs of parameters t_1, t_2 such that $N(q_1(t_1), q_2(t_2), \beta) \cap s \neq \emptyset$. The arrangement of $n - 2$ curves in total can be found in $O(n^2 \lambda_4(n))$ time [6]. Then the difference $[0,1] \times [0,1] \setminus \bigcup_{s \in S \setminus \{s_1, s_2\}} F(s_1, s, s_2)$ can be found in $O(n^2)$ time. Therefore we can verify which edges belong to $G_\beta(S)$ in $O(n^3 \lambda_4(n))$ time. □

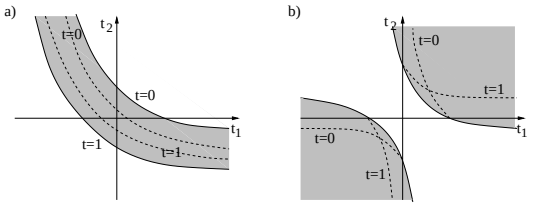


Figure 3: Examples of sets $F(s_1, s, s_2)$ for β near to (a) 0 and (b) 1. Parameter t is considered for clockwise and counterclockwise refractions.

4 Finding β-skeletons for $1 \leq \beta$

According to Theorem 1, for $1 \leq \beta$ we have to consider only $O(n)$ pairs of line segments in S (the pairs corresponding to edges of $DT(S)$). First we consider lune-based β-skeletons. We will analyze pairs of points belonging to given segments $s_1, s_2 \in S$ which generate discs intersected by any segment $s \in S \setminus \{s_1, s_2\}$.

Let $C1(v_1, v_2, \beta)$ be the circle creating the lune $N(v_1, v_2, \beta)$ and containing the point v_1, where $v_1 \in s_1$, $v_2 \in s_2$. Let u be such a point that the segment $v_1 u$ is the diameter of the circle $C1(v_1, v_2, \beta)$. If the line $P(s)$ intersects the arc $N(v_1, v_2, \beta) \cap C1(v_1, v_2, \beta)$ in the point v, then $\angle v_1 v u = \frac{\pi}{2}$. Let h be a homothety with center in v_1 and ratio $\frac{|v_1 v_2|}{|v_1 u|} = \frac{1}{\beta}$ and let line $P'(s)$ be the image of $P(s)$ in this homothety.

Lemma 4 *For a given $\beta \geq 1$, points $v_1 \in P(s_1), v_2 \in P(s_2)$ and the segment $s \in S \setminus \{s_1, s_2\}$, the line $P(s)$ intersects the arc $N(v_1, v_2, \beta) \cap C1(v_1, v_2, \beta)$ in the point v if and only if $\angle v_1 h(v) v_2 = \frac{\pi}{2}$.*

The algorithm computing a lune-based β-skeleton for $\beta \geq 1$ is very similar to the one presented in the previous section. We consider a line $P'(s)$ which is a homothetic image of the line $P(s)$ with ratio $\frac{1}{\beta}$ and center $q_1(t_1)$ (in the next step of the algorithm the same is done for $q_2(t_2)$). The ray shot from $q_1(t_1)$ refracts on $P'(s)$ at a right angle. In this case for a fixed t_1 we analyze only one hyperbola (functions for clockwise and counterclockwise refractions are the same). However, sets $F(s_1, s, s_2)$ and $F(s_2, s, s_1)$ are different (discs creating a lune intersect segment s in different way). Therefore, we have to intersect those sets to obtain a set of pairs of points generating lunes intersected by s.

Theorem 5 *For $\beta \geq 1$ the lune-based β-skeleton $G_\beta(S)$ can be found in $O(n^2 \lambda_4(n))$ time, where $\lambda_4(n)$ denotes the maximum possible length of a $(n, 4)$ Davenport-Schinzel sequence.*

Proof. β-skeletons for $\beta \geq 1$ satisfy inclusions of Theorem 1. Hence, the number of tested edges is linear. For each such pair of segments s_1, s_2 we compute

the corresponding sets of pairs of points generating lunes that do not intersect segments from $S \setminus \{s_1, s_2\}$. Similarly as in Theorem 3 we can do it in $O(n\lambda_4(n))$ time. Therefore, a total time complexity of the algorithm (after analysis of $O(n)$ pairs segments) is $O(n^2\lambda_4(n))$. □

The algorithm for computing circle-based β-skeletons for $\beta \geq 1$ is almost the same as the algorithm for $\beta < 1$.

Theorem 6 *For $\beta \geq 1$ the circle-based β-skeleton $G^c_\beta(S)$ can be found in $O(n^2\alpha(n)\log n)$ time, where $\alpha(n)$ is the inverse Ackermann function.*

Proof. In this case, for any pair $s_1, s_2 \in S$ the number of connected components of the set $[0,1] \times [0,1] \setminus \bigcup_{s \in S \setminus \{s_1,s_2\}} F(s_1, s, s_2)$ is O(n) and for any t_1 there is at most one connected component that contains points with the same t_1 coordinate. For each edge we use Hershberger's algorithm [7] to compute intersection of the complements of sets containing pairs of points generating not empty neighborhoods. We find the lower envelope of curves intersecting upper edge of the square $[0,1] \times [0,1]$ and the upper envelope of curves intersecting lower edge of the square. Then we intersect sets limited by those envelopes. It needs $O(n\alpha(n)\log n)$ time. Hence, the total time complexity of the algorithm is $O(n^2\alpha(n)\log n)$. □

5 Conclusion

The running time of the presented algorithms for β-skeletons for sets of n line segments ranges between $O(n^2\alpha(n)\log n)$ and $O(n^3\lambda_4(n))$ and depends on the value of β. However, we can compute the Gabriel Graph $GG(S)$ in $O(n \log n)$ time. The algorithm relies on the fact that the 2-order Voronoi diagram and the 3-order Voronoi diagram for S can be found in $O(n \log n)$ time [11].

The existence of this algorithm suggests that it may be possible to find a faster way to compute β-skeletons for other values of β, especially for $1 \leq \beta \leq 2$.

The algorithms shown in this work for each pair of segments find such a position of generators that the corresponding lune does not intersect any other segment. We can consider a problem in which we look for an arrangement of all generators satisfying this condition at the same time. Then the method described in the paper can also be used. We analyze n-dimensional space and test if $[0,1]^n \setminus \bigcup_{s_i,s_j \in S, s \in S \setminus \{s_i,s_j\}} F(s_i, s, s_j) \times R^{n-2} \neq \emptyset$, where i and j also define corresponding coordinates in R^n. Unfortunately, such an algorithm is expensive. Is there a more effective algorithm for this problem ? Additional interesting questions about β-skeletons are related to their connections with k-order Voronoi diagrams for line segments.

Acknowledgments

The authors would like to thank Jerzy W. Jaromczyk for important comments.

References

[1] O. Aichholzer, F. Aurenhammer, *Straight Skeletons for General Polygonal Figures in the Plane*, Proceedings of 2nd Computing and Combinatorics Conference (COCOON), 1996, 117-126

[2] M. Brévilliers, N. Chevallier, D. Schmitt, *Triangulations of line segment sets in the plane*, Proceedings of the 27th international Conference on Foundations of Software Technology and Theoretical Computer Science, 2007, 388-399

[3] Ch. Burnikel, K.Mehlhorn, S. Schirra, *How to compute the Voronoi diagram of line segments: Theoretical and experimental results*, Proceedings of 2nd Annual European Symposium On Algorithms (ESA), 1994, 227-239

[4] L.P.Chew, K.Kedem, *Placing the largest similar copy of a convex polygon among polygonal obstacles*, Proceedings of the 5th Annual ACM Symposium on Computational Geometry, 1989, 167-174

[5] D. Eppstein, *β-skeletons have unbounded dilation*, Computational Geometry, vol. 23, 2002, 43-52

[6] J. E. Goodman, J. O'Rourke, *Handbook of Discrete and Computational Geometry*, Chapman & Hall/CRC, 2004

[7] J. Hershberger, *Finding the Upper Envelope of n Line Segments in O(n log n) Time*, Information Processing Letters, 33(4), 1989, 169-174

[8] F. Hurtado, G. Liotta, H. Meijer, *Optimal and suboptimal robust algorithms for proximity graphs*, Computational Geometry: Theory and Applications 25 (1-2), 2003, 35-49.

[9] D.G. Kirkpatrick, J.D. Radke, *A framework for computational morphology*, Computational Geometry, North Holland, Amsterdam, 1985, 217-248

[10] M. Kowaluk, *Planar β-skeleton via point location in monotone subdivision of subset of lunes*, EuroCG 2012, Italy, Assisi, 2012, 225-227

[11] E. Papadopoulou, M. Zavershynskyi, *A Sweepline Algorithm for Higher Order Voronoi Diagrams*, Proceedings of 10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD), 2013, 16-22

[12] K.J. Supowit, *The relative neighborhood graph, with an application to minimum spanning trees*, Journal of the ACM, volume 30, issue 3, 1983, 428-448

Author Index