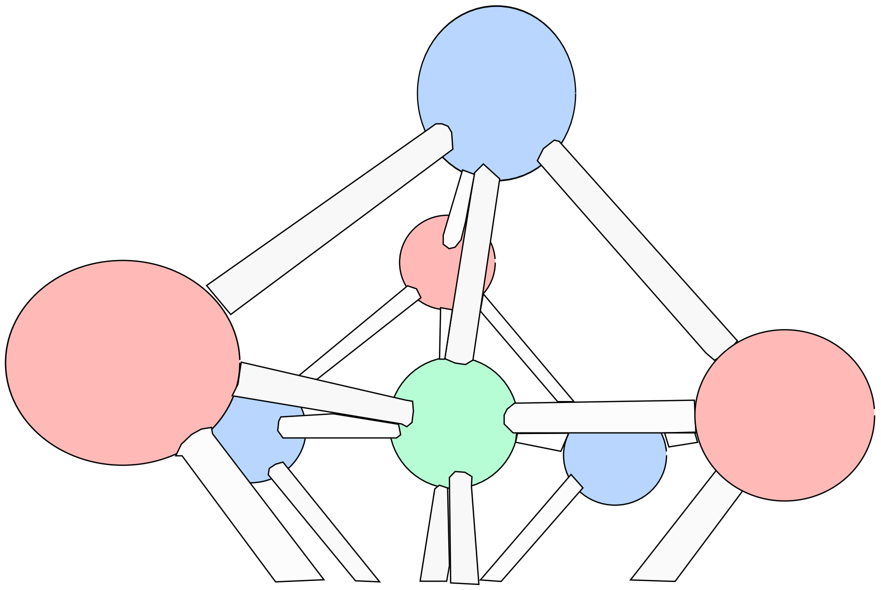


EuroCG'09

25th European Workshop on Computational Geometry

March 16 – 18, 2009 Brussels, Belgium

Contents

Introduction

The 25th European Workshop on Computational Geometry (EuroCG'09) was organized on March 16 – 18 at the Université Libre de Bruxelles (ULB), in Brussels, Belgium. EuroCG is renowned as a friendly conference, where researchers from across Europe and the world can meet and exchange ideas in a stimulating atmosphere.

A record number of 83 papers were presented. The organizers thank all authors and invited speakers for their participation, as well as the 51 reviewers for their insightful comments. We also wish to thank the sponsors, the Fonds National de la Recherche Scientifique (F.R.S.-FNRS) and Emakina.

More information about this conference and about previous and future editions is available online at

`http://eurocg.org`

Invited Speakers

- Prosenjit Bose, Carleton U.
- Luc Devroye, McGill U.
- Mark Overmars, Utrecht U.

Organizing Committee

- Greg Aloupis, ULB
- Jean Cardinal, ULB
- Sébastien Collette, ULB
- Gwenaël Joret, ULB
- Stefan Langerman, ULB (chair)
- Benoit Macq, UCL

Program Committee

- Greg Aloupis, ULB
- Jean Cardinal, ULB
- Sébastien Collette, ULB
- Erik Demaine, MIT
- Marcin Kaminski, ULB
- Stefan Langerman, ULB (chair)
- Pat Morin, Carleton U.
- Perouz Taslakian, McGill U.
- Stefanie Wuhrer, Carleton U.

Other Reviewers

- Luca Castelli Aleardi
- Zouhour Ben Azouz
- Eric Berberich
- Alan Brunton
- Paz Carmi
- Gail Carmichael
- Daniele Catanzaro
- Jean-Paul Doignon
- Karim Douïeb
- Vida Dujmović
- Céline Engelbeen
- Mohammad Farshi
- Rosa Maria Videira de Figueiredo
- Samuel Fiorini
- Delia Garijo
- John Goldak
- Meng He
- Pim van 't Hof
- Raphaël Jungers
- Matias Korman
- Adrian Kosowski
- Hugo Ledoux
- Martin Milanič
- Dieter Mitsche
- Yonatan Myers
- Catharina Olsen
- Belén Palop
- Michel Paquette
- Daniel Paulusma
- Vincent Pilaud
- Daniel Raible
- Pedro Ramos
- Daria Schymura
- Rodrigo Silveira
- Michiel Smid
- Shakhar Smorodinsky
- Costas Tsirogiannis
- George Tzoumas
- Alper Ungor
- Eric Colin de Verdière
- Pengcheng Xi
- Pawel Zylinski

Probabilistic Analysis of Algorithms

Luc Devroye[*]

Abstract

The theme of this expository talk is the interaction between classical probability theory and the analysis of algorithms. What are the researchers in the area of "probabilistic analysis of algorithms" studying nowadays? What are the main tools that have driven progress? For example, one of the oldest randomized algorithms, Quicksort, has been asymptotically cracked by the clever use of stochastic fixed-point equations. The corresponding search structure, the random binary search tree, and many related random trees, are best approached via branching processes. Random structures of many genetic origins can be understood by showing their closeness to well-understood (often continuous) random processes. The talk will revisit several of the success stories in the field.

[*]School of Computer Science, McGill University, Montreal, Canada, `lucdevroye@gmail.com`

Witness Gabriel Graphs[*]

Boris Aronov[†] Muriel Dulieu[†] Ferran Hurtado[‡]

Abstract

We consider a generalization of the Gabriel graph, the *witness Gabriel graph*, and study several of its properties, both as a proximity graph and as a new tool in graph drawing.

1 Introduction

Originally defined to capture some concept of neighborliness, *proximity graphs* [7,10,13] can be intuitively defined as follows: given a set P of points in the plane, the vertices of the graph, there is an edge between a pair of vertices $p, q \in P$ if some specified region in which they interact contains no point from P, besides possibly p and q.

Proximity graphs have proved to be a very useful tool in shape analysis and in data mining [7,14]. In graph drawing, a problem that has been attracting substantial research is to explore which classes of graphs admit a proximity drawing, for some notion of proximity, and when it is possible to efficiently decide, for a given graph, whether such a drawing is possible [2,10]. For all these reasons, several variations and extensions have been considered, from higher-order proximity graphs to the so called weak proximity drawings [3,7].

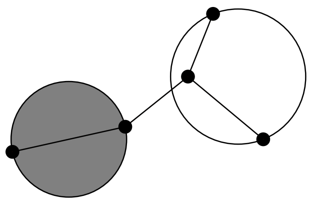


Figure 1: Gabriel graph. The vertices defining the shaded disk are adjacent because their disk doesn't contain any other vertex, in contrast to the vertices defining the white disk.

In the case of the *Gabriel graph*, GG(P), the region of influence of a pair of vertices a, b is the closed disk with diameter ab, D_{ab}. An edge ab is in the Gabriel graph of a point set P if and only if $P \cap D_{ab} = \{a, b\}$ (see figure 1). Gabriel graphs were introduced by Gabriel and Sokal [6] in the context of geographic variation analysis.

We consider in this work a generalization of the Gabriel graph, the *witness Gabriel graph*, GG(P, Q). It is defined by two sets of points P and Q; P is the set of vertices of the graph and Q is the set of *witnesses*. There is an edge ab in GG(P, Q) if, and only if, there is no point of Q in $D_{ab} \setminus \{a, b\}$ (see figure 2).

Notice that witness Gabriel graphs are a proper generalization of Gabriel graphs, because when the set Q of witnesses coincides with the set P of vertices, we clearly obtain GG$(P, P) =$ GG(P). This was the main underlying idea of the generic concept of *witness graphs*, which were introduced as a general framework in [1] to provide a generalization of proximity graphs, allowing witnesses to play a negative role as in this paper or the opposite as well. Several examples were described in [1], including in particular Delaunay graphs and rectangle-of-influence drawings. A systematic study is developed in [5].

In this paper we prove several fundamental properties of witness Gabriel graphs, describe algorithms for their computation, and present results on the realizability of some combinatorial graphs.

We assume throughout the paper that the points in $P \cup Q$ are in general position, that is, there are no three points on a line in $P \cup Q$ and no four on a circle.

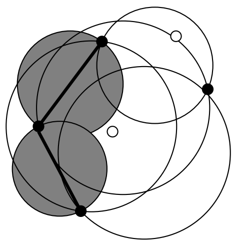


Figure 2: GG(P, Q). Black points are the vertices of the graph, white points are the witnesses. Each pair of vertices defining a grey disk are adjacent as opposed to pair of points defining a white disk.

2 Some Properties of Witness Gabriel Graphs

It is known that that MST$(P) \subseteq$ GG$(P) \subseteq$ DT(P) [8], where MST(P) is the minimum spanning tree and DT(P) is the Delaunay triangulation. As a con-

[*]Part of this research was done while the third author was visiting Brooklyn Polytechnic supported by Grant AGAUR-Generalitat de Catalunya 2007 BE-1 00033.

[†]Polytechnic Institute of NYU. Partially supported by a grant from the U.S.-Israel Binational Science Foundation, by NSA MSP Grant H98230-06-1-0016, and NSF Grant CCF-08-30691.

[‡]Universitat Politècnica de Catalunya. Partially supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

sequence, $|\,\mathrm{MST}(P)| \leq |\,\mathrm{GG}(P)| \leq |\,\mathrm{DT}(P)|$. Expressing this in terms of $n = |P|$, we have that $n-1 \leq |\,\mathrm{GG}(P)| \leq 3n-6$. In [12], a more detailed analysis gives a tighter upper bound of $3n-8$.

For witness Gabriel graphs $\mathrm{GG}(P,Q)$, the situation is quite different, as for any fixed set P of n points, by varying the size of Q and the location of the witnesses, the number of edges in $\mathrm{GG}(P,Q)$ can attain any value from 0 to $\binom{n}{2}$.

For example, when $Q = \varnothing$, we obviously obtain $\mathrm{GG}(P,\varnothing) = K_n$. The reverse problem is more interesting: as the witness points can be thought as *interferences* that prevent the points in P from interacting, one may wonder how many witnesses are required to completely eliminate the interactions in P. Trivially, if there is a witness inside each disk D_{ab}, for all $a, b \in P$, then $\mathrm{GG}(P,Q)$ has no edge, and this can be achieved by putting a witness close to the midpoint of every pair a, b of points from P, which would give $|Q| \leq \binom{n}{2}$. In the following theorem we present a much better bound for the number of witnesses necessary to eliminate all edges of $\mathrm{GG}(P,Q)$.

Theorem 1 *$n-1$ witnesses are always sufficient to eliminate all edges of an n-vertex witness Gabriel graph, while $\frac{3}{4}n - o(n)$ witnesses are sometimes necessary.*

According to the preceding result, $n-1$ suitable witnesses can always destroy all the edges. Interestingly, realizing some witness Gabriel graphs may require strictly more witnesses:

Theorem 2 *There exist witness Gabriel graphs on n vertices, for which at least $\frac{3}{2}n - 8$ witnesses are necessary.*

3 Witness Gabriel Drawings

Given a combinatorial graph $G(V,E)$, a *witness Gabriel drawing* of G is a mapping of its vertex set V to a set P of distinct points in the plane, together with a set of witnesses Q, such that the mapping induces an isomorphism between $G(V,E)$ and $\mathrm{GG}(P,Q)$. The fundamental question concerning witness Gabriel drawability is the following: Given a graph G, is it possible to construct a witness Gabriel drawing of G? That question has been studied for (standard) Gabriel graphs. In [4] Bose et al. present a complete characterization of those trees that are drawable as a Gabriel graph. They proved that such trees cannot have vertices of degree greater than four and cannot have two adjacent vertices of degree four. They also characterized Gabriel-drawable trees by exhibiting families of forbidden subtrees and by showing that they don't contain members of these families.

Lubiw and Sleumer [11] showed that all maximal outerplanar graphs admit a Gabriel drawing in the plane. They also conjectured that any biconnected outerplanar graph has a Gabriel drawing. This was settled in the affirmative by Lenhart and Liotta [9].

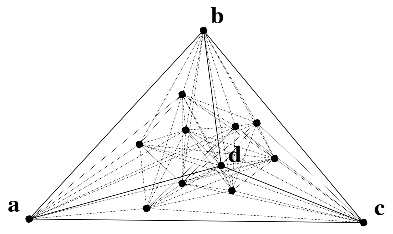


Figure 3: All the vertices inside the triangle $\triangle abc$ will be connected to d.

As every Gabriel graph is also a witness Gabriel graph, since $\mathrm{GG}(P) = \mathrm{GG}(P,P)$, one may expect more power of representation with the use of the witnesses. This is the case for tress, a class of graphs we have studied with witness Gabriel graphs, for which we have found the following:

Theorem 3 *For all trees, a witness Gabriel drawing exists.*

We first observe some properties of witness Gabriel drawings, before deducing that some graphs have no such drawings.

Lemma 4 *In a witness Gabriel graph, for any pair of incident edges ab and bc, all the points $p \in P$ in the triangle $\triangle abc$ are connected to the common vertex b, and there is no witness in $\triangle abc$.*

Proposition 1 *If a witness Gabriel graph has as a subgraph a triangle $\triangle abc$, the vertices a, b, c and the vertices inside this triangle form a complete subgraph (see figure 3).*

We denote by $K_{i,i,i}$ the complete 3-partite graph in which each part contains i vertices; we associate a color to each part; similarly, $K_{i,i,i,i}$ denotes the complete 4-partite graph.

Lemma 5 *It is impossible, in a witness Gabriel drawing of $K_{2,2,2}$ or $K_{2,2,2,2}$, for a line containing two vertices of one color, to divide the plane in two half-planes, each of them containing two points of a second and third color respectively (see figure 4). The second and the third color may be the same.*

Lemma 6 *Any witness Gabriel drawing of $K_{2,2,2,2}$ must have vertices in convex position.*

Lemma 7 *There is no witness Gabriel drawing of $K_{2,2,2,2}$ in which all the vertices of the same color are consecutive (see figure 5).*

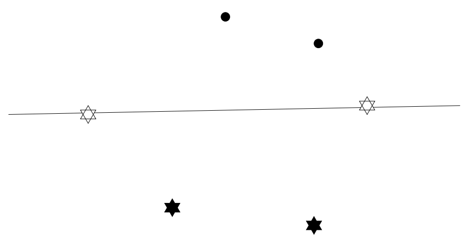


Figure 4: The line containing the white-star points separates two black-dot points from two black-star points.

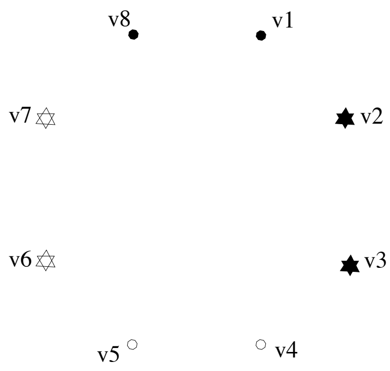


Figure 5: $K_{2,2,2,2}$ in which all the vertices of the same kind are consecutive.

The constraints described in the preceding results lead to a graph that is not drawable:

Theorem 8 *There is no witness Gabriel drawing of* $K_{3,3,3,3}$.

From the preceding result we immediately obtain the following:

Corollary 9 *No graph containing* $K_{3,3,3,3}$ *as an induced subgraph can be drawn as a witness Gabriel graph. In particular, there is no witness Gabriel drawing of* $K_{p,q,r,s}$ *for* $p, q, r, s \geq 3$.

4 Construction Algorithms

In this section we describe two algorithms to compute the witness Gabriel graph $\mathrm{GG}(P, Q)$ from two given sets of points P and Q.

Theorem 10 *Given two point sets* P, Q *with* $|P| + |Q| = n$, *the graph* $\mathrm{GG}(P, Q)$ *can be computed in* $O(n^2)$ *time, which is worst-case optimal.*

First algorithm: For each point $p \in P$, do the following: For each point $q \in Q$, draw a line l_q through q, perpendicular to the line pq. Consider the intersection I_p of the half-planes containing p bounded by the lines l_q, $\forall q \in Q$. Then, the edge pr, $r \in P \setminus \{p\}$, is in $\mathrm{GG}(P, Q)$ if and only if $r \in I_p$ (see figure 6). Once

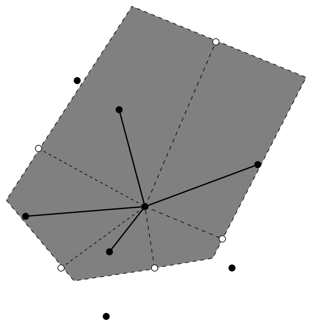


Figure 6: The first algorithm to build $\mathrm{GG}(P, Q)$. Black points are in P and white points in Q.

we have computed the circular ordering of points in $P \cup Q$ around p, we can compute I_p and identify all edges pr in linear time.

Second algorithm: Build the Voronoi diagram $\mathrm{Vor}(Q)$ of the points in Q. For each $p \in P$, add the point p to $\mathrm{Vor}(Q)$ and consider all segments of the form pr with $r \in P \setminus \{p\}$. For each edge pr take the midpoint $m(p, r)$. Observe that $m(p, r)$ is in the Voronoi cell of p in $\mathrm{Vor}(Q \cup \{p\})$ if and only if the edge pr is in $\mathrm{GG}(P, Q)$. The algorithm can be implemented to run in quadratic time using standard tools. Again, it is useful to have the circular ordering of all points in $P \cup Q$ around each point in P.

5 Verification Algorithm

In this section we present an algorithm to verify whether a graph $G(V, E)$ embedded in the plane can be a witness Gabriel graph $\mathrm{GG}(V, Q)$, for some suitable set of witnesses Q.

Theorem 11 *Given a straight-line graph* $G = G(V, E)$ *embedded in the plane, checking if there exists a set of witnesses Q so that G coincides with* $\mathrm{GG}(V, Q)$ *can be done in* $O(|V|^2 \log |E|)$ *time; if the answer is positive, such a set of witnesses Q can be computed within the same time.*

Algorithm For each edge pq in G, draw a disk D_{pq} with diameter pq. Take the union $U = \bigcup_{p,q \in G} D_{pq}$ of these disks. Draw the Voronoi Diagram of the arcs and vertices of the boundary of U. For each pair of vertices r and s such that there is no edge between them in G, draw a disk D_{rs} with diameter rs. Check if the center c of D_{rs} lies in U. If it does, find which cell C of the Voronoi diagram contains c and check if the site of C intersects D_{rs}, if it doesn't, $D_{rs} \subset U$, it is impossible to place a witness to eliminate rs without also eliminating a legitimate edge of G. Therefore G is not a witness Gabriel drawing $\mathrm{GG}(V, Q)$, for any Q (see figure 7).

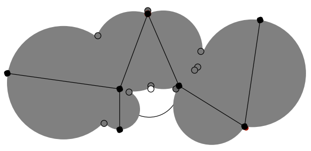


Figure 7: The algorithm to check if a geometric graph is a witness Gabriel drawing. Black points are in P.

6 Final Remarks

There are some obvious open problems left in this paper, like closing the gaps between some bounds.

On the other hand in this version we have omitted the description of some extensions. For example, the same way that standard Gabriel graphs are extended to higher order, this can be done for the witness generalization: In a witness k-Gabriel graph, an edge ab, $a, b \in P$, is in the graph if there are fewer than k witnesses in $D_{ab} \setminus \{a, b\}$. Most of the preceding results can be easily modified to provide the corresponding conclusions about witness k-Gabriel graphs.

Finally, let us also mention that some natural long term goals, such as a complete characterization of the class of witness Gabriel graphs or the design of efficient algorithms testing graphs for membership, remain elusive to date, which, on the other hand, is a common situation for most classes of standard proximity graphs

References

[1] B. Aronov, M. Dulieu, and F. Hurtado. Witness graphs. *Manuscript*, 2008.

[2] G. Di Battista, Peter Eades, and R. Tamassia an I. G. Tollis. *Graph Drawing. Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.

[3] G. Di Battista, G. Liotta, and S. Whitesides:. The strength of weak proximity. *Journal of Discrete Algorithms*, 4(3):384–400, 1992.

[4] P. Bose, W. Lenhart, and G. Liotta. Characterizing proximity trees. In G. Di Battista, P. Eades, H. de Fraysseix, P. Rosenstiehl, and R. Tamassia, editors, *Proc. ALCOM Int. Work. Graph Drawing, GD*, pages 9–11. Centre D'Analyse et de Mathématique Sociales, Paris Sorbonne, 1993.

[5] M. Dulieu. Ph d thesis, in preparation.

[6] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18:259–278, 1969.

[7] J.W. Jaromczyk and G.T. Toussaint. Relative neighborhood graphs and their relatives. *P-IEEE*, 80:1502–1517, 1992.

[8] D.J. Kirkpatrick and J.D. Radke. A framework for computational morphology. *Computational Geometry*, 85:217–248, 1985.

[9] W. Lenhart and G. Liotta. Proximity drawings of outerplanar graphs. In *Graph Drawing*, pages 286–302, 1996.

[10] G. Liotta. Proximity drawings. In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization.* CRC Press, to appear.

[11] A. Lubiw and N. Sleumer. Maximal outerplanar graphs are relative neighborhood graphs. *Proc. 5th Canad. Conf. Comput. Geom.*, pages 198–203, 1993.

[12] D.W. Matula and R.R. Sokal. Properties of Gabriel graphs relevant to geographical variation research and the clustering of points in the plane. *Geographical analysis*, 12:205–222, 1980.

[13] G. Toussaint. Some unsolved problems on proximity graphs, 1991.

[14] G.T. Toussaint. Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining. *International Journal of Comput. Geom. and Applications*, 15(2):101–150, 2005.

Measuring the Similarity of Geometric Graphs*

Otfried Cheong[†] Joachim Gudmundsson[‡] Hyo-Sil Kim[†] Daria Schymura[§] Fabian Stehn[§]

Abstract

What does it mean for two geometric graphs to be similar? We propose a distance for geometric graphs that we show to be a metric, and that can be computed by solving an integer linear program. We also present experiments using a heuristic distance function.

1 Introduction

Computational geometry has studied the matching and analysis of geometric shapes from a theoretical perspective, and developed efficient algorithms measuring the *similarity* of geometric objects. Two objects are similar if they do not differ much geometrically. A survey by Alt and Guibas [1] describes the significant body of results obtained by researchers in computational geometry in this area.

This paradigm fits a number of practical shape matching problems quite well, such as the recognition of symmetries in molecules, or the self-alignment of a satellite based on star patterns. Other pattern recognition problems, however, seem to require a different definition of "matching." For instance, recognizing logos, Egyptian hieroglyphics, Chinese characters, or electronic components in a circuit diagram are typical examples where this is the case. The same "pattern" can appear in a variety of shapes that differ geometrically. What remains invariant, however, is the "combinatorial" structure of the pattern.

We propose to consider such patterns as geometric graphs, that is, planar graphs embedded into the plane with straight edges. Two geometric graphs can be considered similar if both the underlying graph and the geometry of the planar embedding are "similar." The distance measures considered in computational geometry, such as the Hausdorff distance, Fréchet distance, or the symmetric difference, do not seem to apply to geometric graphs.

Pattern recognition systems that combine a combinatorial component with a geometric component are already used in practice—in fact, *syntactic* or *structural* pattern recognition is based on exactly this idea: A syntactic recognizer decomposes the pattern into geometric primitives and makes conclusions based on the appearance and relative position of these primitives [2, 6]. While attractive from a theoretical point of view, syntactic recognizers have not been able to compete with numerical or AI techniques for character recognition [5]. In general, the pattern recognition community may be said to consider graph representations as expressive, but too time-consuming, as subgraph isomorphism in general is intractable.

An established measure of similarity between (labeled) graphs is the *edit distance*. The idea of an edit distance is very intuitive: To measure the difference between two objects, measure how much one object has to be changed to be transformed into the other object. To define an edit distance, one therefore defines a set of allowed operations, each associated with a cost. An edit sequence from object A to object B is a finite sequence of allowed operations that transforms A into B. The distance between A and B is the minimal cost of an edit sequence from A to B.

The edit distance originally stems from string matching where the allowed operations are insertion, deletion and substitution of characters. The edit distance of strings can be computed efficiently, and the string edit distance is used widely, for instance in computational biology.

Justice and Hero [4] give a definition of an edit distance for vertex-labeled graphs that additionally allows relabeling of vertices, and give an integer linear programming formulation. The edit operations are insertion and deletion of vertices, insertion and deletion of edges, and a change of a vertex label.

It is natural to try to define an edit distance for geometric graphs as well. Simply considering a geometric graph as a graph whose vertices are labeled with their coordinates is not sufficient, as the cost of inserting and deleting an edge should also be dependent on the length of the edge. This leads to the following operations: Insertions and deletions of vertices, translations of vertices, and insertions and deletions of edges. However, it is difficult to give bounds on the length of an edit sequence: vertices can move several times to make insertions and deletions cheaper.

This leads us to define another graph distance function, given in the next section. It is not an edit distance, and so we need to prove explicitly that it is a metric. We also give an integer linear programming formulation that allows us to compute our distance for

*This work was supported by Korea Science & Engineering Foundation through the Joint Research Program (F01-2006-000-10257-0).

[†]KAIST, Korea. {otfried,hyosil}@tclab.kaist.ac.kr
[‡]NICTA, Australia. joachim.gudmundsson@nicta.com.au
[§]FU Berlin, Germany. {schymura,stehn}@inf.fu-berlin.de

small graphs with an ILP solver. Unfortunately, we do not know how to compute or even approximate our graph distance for larger graphs. In fact, we give two reductions from NP-hard problems, but both result in non-"practical" instances of the problem.

We therefore turn our attention to a heuristic. We define the *landmark distance* of two geometric graphs, and present pattern retrieval experiments on a database of 25056 graphs created from glyphs of Chinese characters. The idea of the landmark distance is to represent a geometric graph on n vertices as a set of n points in $\mathbb{R}^6$. The landmark distance between two geometric graphs is then the Earth Mover's Distance between the point sets representing the graphs.

2 Geometric graph distance

Our distance is inspired by the graph edit distance: it is based on primitive operations, namely insertion and deletion of vertices and edges, and the translation of vertices. However, we do not allow arbitrary sequences of these operations. Instead the edit operations must be performed in this order:

1. Edge deletions
2. Vertex deletions
3. Vertex translations
4. Vertex insertions
5. Edge insertions

Vertex insertions and deletions are free, insertion or deletion of an edge e of length $\ell(e)$ have cost $C_E \cdot \ell(e)$, and translating a vertex has cost C_V times the distance of the translation plus C_E times the change in the length of the incident edges. Note that we measure the change in edge length from graph A to graph B, and not for the individual operations.

Taking into account the change in edge length is necessary, as otherwise the distance would not satisfy the triangle inequality, see Figure 1.

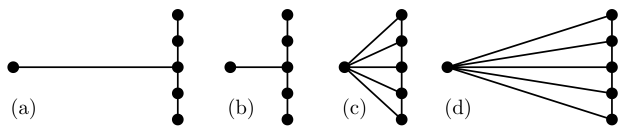


Figure 1: If the change in edge lengths are not taken into consideration then it is cheaper to go from graph (a) to graph (d) via graphs (b) and (c).

We can now show:

Theorem 1 *The geometric graph distance defined above is a metric on the set of geometric graphs without isolated vertices.*

The geometric graph distance can be formulated as an ILP as follows. Let $A = (V(A), E(A))$ and $B = (V(B), E(B))$ be the two geometric graphs. For each vertex v_i of $V(A)$ and v_j in $V(B)$, we have a binary variable V_{ij} which is 1 if v_i is moved to v_j, otherwise 0. Similarly, for each edge e_i of $E(A)$ and e_j in $E(B)$, we have a binary variable E_{ij} which is 1 if the two endpoints of e_i are moved to the endpoints of e_j, otherwise 0. Furthermore, each vertex can only be moved once, hence, $\forall i \sum_j V_{ij} \leq 1$ and $\forall j \sum_i V_{ij} \leq 1$.

An edge can only be matched if the endpoints are the same, i.e., if $e_i = (v_a, v_b)$ and $e_j = (v_p, v_q)$ then $E_{ij} \leq \frac{1}{2} \cdot (V_{ap} + V_{bq} + V_{aq} + V_{bp})$.

These are all the constraints. The distance is the minimum of the function:

$$C_V \cdot \sum_{i,j}(|v_i v_j| \cdot V_{ij}) + C_E \cdot \Big(\sum_i \ell(e_i) + \sum_j \ell(e_j) \Big)$$
$$-C_E \cdot E_{ij} \cdot \sum_{i,j} \Big(\ell(e_i) + \ell(e_j) - \big|\ell(e_i) - \ell(e_j)\big| \Big).$$

The first term is the cost of moving all vertices (the remaining vertices are deleted and inserted, which is for free). The second and third terms are the total cost of first deleting all edges of A and then inserting all edges of B. The only way to avoid deleting and inserting an edge is by moving it from A to B. In that case, E_{ij} is 1, and the cost is the difference in edge length, which is modeled by the fourth term.

Theorem 2 *The problem of computing the geometric graph distance as defined above is NP-hard.*

The theorem can be proven by using a reduction from the 3D-matching problem or the Hamiltonian path problem. However, we can only prove the theorem in the case when the two graphs are highly non-planar or for planar graphs when $C_V \ll C_E$. Thus, both results are for non-"practical" instances of the problem

3 Landmark Distance

The idea of the landmark distance is to designate a few vertices as *landmarks* and to represent the vertices of the graph by their distances to the landmarks.

Formally, let $G = (V, E)$ be a geometric graph, and let $C(G)$ be the complete graph on the set V. An edge (u, v) of $C(G)$ is given a weight as follows: if $(u, v) \in E$, then its weight is $|uv|$, otherwise its weight is $p \cdot |uv|$, where $p > 1$ is a fixed *penalty* value. The distance $d_G(u, w)$ between a vertex $u \in V$ and a landmark $w \in V$ is then defined as the length of the shortest path between u and w in $C(G)$. After testing different values for p we chose $p = 1.6$ for our experiments.

Let $L = w_1, \ldots, w_k$ be k vertices of G called *landmarks.* For a vertex $v \in V$ with coordinates (x, y), we define the *L-vector* L_v of v as the $(k+2)$-dimensional vector containing the distances of v to the k landmarks as well as the coordinates of v:

$$L_v = (d_G(v, w_1), \ldots, d_G(v, w_k), x, y).$$

The *landmark representation* $R(G)$ of a graph G is now simply the set of L-vectors of all vertices: $R(G) := \{L_v \mid v \in V(G)\}$.

We define the *landmark distance* $d_L(G_0, G_1)$ between two geometric graphs G_0 and G_1 (both with given landmarks) as the *normalized Earth Mover's Distance* between the point sets $R(G_0)$ and $R(G_1)$.

The normalized Earth Mover's Distance (nEMD) is a distance measure defined on weighted point sets [7]. Let P and Q be two weighted point sets with $\sum_{p \in P} w(p) = \sum_{q \in Q} w(q) = 1$, where $w(u) \geq 0$ is the weight of a point u.

Intuitively, the set P can be seen as a set of earth piles and the set Q can be seen as holes in the ground, where the weight of each point stands for the amount of earth at that position or the size of the hole, respectively. The Earth Mover's Distance is then defined as the *cheapest* way to move the earth into the holes, where piles can be split and the cost of transporting s units of earth from a pile to a hole is equal to s times the distance between the pile and the hole.

The nEMD can be formalized by the following linear program. Let d_{pq} denote the Euclidean distance of $p \in P$ to $q \in Q$ and let f_{pq} denote the flow from p to q. Then the nEMD(P, Q) is defined as the minimal total cost that establish a maximum flow of 1:

$$\text{nEMD}(P, Q) = \min \sum_{p \in P} \sum_{q \in Q} f_{pq} d_{pq}$$

subject to the following constraints

$$\begin{aligned} \forall p \in P \; \forall q \in Q \; f_{pq} &\geq 0 \\ \forall p \in P \; \sum_{q \in Q} f_{pq} &= w(p) \\ \forall q \in Q \; \sum_{p \in P} f_{pq} &= w(q) \end{aligned}$$

In the definition of the landmark distance, we give all vertices equal weight and use the ℓ_1-metric in $\mathbb{R}^{k+2}$ as the underlying distance for the nEMD.

Theorem 3 *The landmark distance on graphs with given landmarks has the following properties:*
(i) $d_L(G_0, G_1) \geq 0$,
(ii) $d_L(G_0, G_1) = d_L(G_1, G_0)$,
(iii) $d_L(G_0, G_2) \leq d_L(G_0, G_1) + d_L(G_1, G_2)$.

All the properties in the theorem follow directly from the fact that the normalized EMD is a metric [7]. Note that $d_L(G_0, G_1) = 0$ does not imply $G_0 = G_1$.

4 Experimental results

We performed pattern retrieval experiments on a database of graphs generated from Chinese character glyphs using the landmark distance. The motivation here was not to build a Chinese character recognition system—a lot of research has been done in this area, and it is not our intention to compete with these finely tuned results of years of research. We turned to Chinese characters because we found them to be a source of large number of graphs with known semantics.

We selected six different fonts, including two Korean, two Japanese, and two Chinese fonts. We picked a set of 4176 Chinese characters (or, more precisely, Unicode code points) that exist in all six fonts, and generated graphs for each of these $6 \times 4176 = 25056$ glyphs as follows: We draw the glyph and compute its medial axis. We prune away small features, and then simplify each chain of degree-two vertices using the Imai-Iri algorithm [3]. Figure 2 shows three examples of glyphs and the corresponding graphs. For the distance computations, all graphs were then linearly scaled to fill a unit square (not shown in the figure).

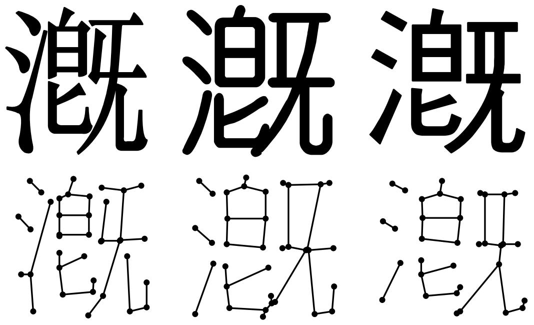


Figure 2: Three glyphs and generated graphs for U+6f11 "slowly flowing water".

As explained, our database consists of 4176 sets of six graphs that represent the same abstract Chinese character. In principle, the graphs representing the same character should be similar, and we have assumed this as the ground truth of our database. In reality, there can be considerable variation between graphs with the same semantics, due to a different glyph style, or actual variations in character shape.

The experiment We selected one of the six fonts (the Korean "dotum" font) as our reference font, and built a database of 4176 "model" graphs from this.

We then considered each of the remaining 20880 "pattern" graphs, and computed its distance to each model, using the EMD implementation by Rubner et al. [7]. The models were then sorted in order of distance from the pattern. Ideally, the nearest model should be a graph for the same Chinese character, so we determined the index of the occurrence of the same Chinese character in the ranked list of models.

As a control experiment, we first tried this experiment using the Hausdorff distance of the graph vertices. The results are given in the first line of Table 1: For 31.8% of the 20880 patterns the best (most similar) model belonged to the same Chinese character, for 50.9% of the patterns, one of the first (most similar) ten models belongs to the same character.

It is clear that the Hausdorff distance does not capture the problem well enough (even though we ignore edge information here, we do not believe that using the edge information would actually help).

The Earth Mover's Distance, however, does much

Graph distance		Index in ranked model list							
		1	2	3	4	5	6–10	11–20	21–200
Hausdorff distance	# patterns	6647	1272	705	496	387	1115	1257	4283
of vertices	accum %	31.8	37.9	41.3	43.7	45.5	50.9	56.9	77.4
EMD	# patterns	16844	1646	519	258	184	286	193	200
of vertices	accum %	80.7	88.6	91.0	92.3	93.2	94.5	95.4	96.4
Landmark	# patterns	17814	1298	454	260	152	317	195	221
distance	accum %	85.3	91.5	93.7	95.0	95.7	97.2	98.1	99.2

Table 1: A summary of our experimental studies of Chinese character retrieval.

better, even when we ignore all information about the edges of the graph. The second line of our table shows this experiment, where we ranked the models by nEMD of the graph vertices only. The results are surprisingly good considering that the edge information is not used at all: for 94.5% of the patterns, the correct Chinese character is found in the top ten.

Landmark selection Selecting the right landmarks in each graph is critical to the success of the landmark distance. We fixed four landmarks in each model graph, by choosing the four vertices that are extreme in the four diagonal directions. For a pattern, we actually try all plausible choices of landmarks, and use the landmarks that result in the smallest distance to a given model (so the choice of landmarks could be different for each model).

Speeding up the computation It turned out that computing the EMD is rather expensive, and computing 4176×20880 EMD distances in every experiment is very time-consuming.

We therefore used a heuristic to speed up the computation: Instead of using the EMD, we compute a simplified landmark distance $d'_L(G_0, G_1)$ which is defined as follows: For each point u in $R(G_0)$, find the nearest point $nn(u) \in R(G_1)$. Then

$$d'_L(G_0, G_1) := \sum_{u \in R(G_0)} ||u - nn(u)||_1.$$

We first rank all models using this simplified landmark distance. We then look at the nearest 200 models, and recompute the distance from the pattern to these 200 models using the landmark distance.

The heuristic greatly speeds up the computation while having little effect on the quality of the recognition. As an example of the speed-up we compared one character (6f01s) against the entire database which required 31 seconds using the EMD while the above heuristic only required 1.8 seconds, which is a speed-up of approximately 17. However, it should be noted that the exact timings depend very much on the character.

For 85.3% of the patterns this approach finds the same Chinese character and in 97.2% of the cases it is in the top ten.

5 Conclusions and open problems

We believe that we have only scratched the surface of this problem. We gave some evidence that our geometric graph distance is hard to compute, but we lack a formal proof that it is NP-hard for planar graphs with realistic values of C_E and C_V.

Is there a PTAS for our distance, or at least a constant-factor approximation?

If we do not want insertions and deletions of vertices to be free, can we incorporate that into our distance?

Finally, a major problem of our metric is that it does not allow us to cheaply "bend" an edge, that is, to insert a degree-two vertex into an edge and then to move that vertex slightly. Can we define a metric that allows this while still being computable at least through integer linear programming?

References

[1] H. Alt and L.J. Guibas. *Discrete Geometric Shapes: Matching, Interpolation, and Approximation*, pages 121–153. Elsevier B.V., 2000.

[2] K. S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[3] H. Imai and M. Iri. Polygonal approximations of a curve - formulations and algorithms. In *Computational Morphology*, pages 71–86. Elsevier B.V., 1988.

[4] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, 2006.

[5] S. Lucas, E. Vidal, A. Amiri, S. Hanlon, and J. C. Amengual. A comparison of syntactic and statistical techniques for off-line OCR. In *2nd International Colloquium Grammatical Inference and Applications*, pages 168–179. Springer-Verlag, 1994.

[6] T. Pavlidis. *Structural Pattern Recognition*. Springer-Verlag, New York, 1977.

[7] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth movers distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2), 2000. `http://robotics.stanford.edu/~rubner`.

Spanner Properties of $\pi/2$-Angle Yao Graphs [*]

Mirela Damian [†] Nawar Molla [‡] Val Pinciu[§]

Abstract

We show that the YaoYao graph YY_4 is not a spanner. We conjecture that the Yao graph Y_4 is a spanner, and prove this conjecture for the special case of points in convex position.

1 Introduction

Let V be a set of n points in the plane and let $G = (V, E)$ be the complete Euclidean graph on V. For any pair of vertices $u, v \in V$, we use $\delta_G(u, v)$ to denote the length of a shortest path from u to v in G. For fixed $t \geq 1$, H is called a *t-spanner* for G if, for any two vertices $u, v \in V$, $\delta_H(u, v) \leq t \cdot |uv|$. The value t is called the *stretch factor* of H. If t is constant, then H is called a *length spanner*, or simply a *spanner*.

The *Yao graph* [3] with an integer parameter $k > 0$, denoted Y_k, is defined as follows. At each node $u \in V$, any k equal-separated rays originated at u define k cones. In each cone, pick a shortest edge uv, if there is one, and add to Y_k the directed edge $\overrightarrow{uv}$. Ties are broken arbitrarily. It was shown that, for $k > 6$, the Yao graph is a spanner with stretch factor $\frac{1}{1-2\sin\pi/k}$, but it can have degree as high as $n - 1$. To bound the degree, Li et al. [2] proposed another structure called *YaoYao* YY_k, which is constructed by applying a reverse Yao structure on Y_k: at each node u in Y_k, discard all directed edges $\overrightarrow{vu}$ from each cone centered at u, except for a shortest one (again, ties can be broken arbitrarily). YY_k has maximum node degree $2k$, a constant. However, the tradeoff between degree and spanner property is unclear in that the question of whether YY_k is a spanner or not remains open.

The class of graphs Y_k, for $k < 6$, has not been much studied. As far as we know, the only existing result shows that Y_4 is a *weak spanner* with stretch factor $t' = \sqrt{3 + \sqrt{5}}$ (Thm. 2.2, [1]), which means the following: for any two points $u, v \in V$, there is a directed path P from u to v in Y_4 such that, for each point $w \in P$, $|wv| \leq t' \cdot |uv|$. In other words, the entire path P lies inside a ball of radius $t' \cdot |uv|$ centered at v. For any path P and any pair of vertices u and v on P, let $P[u, v]$ denote the subpath of P that extends between u and v.

In this paper we study the class of graphs Y_4 and YY_4. We show that YY_4 is not a spanner, and Y_4 is a spanner for points in convex position. We conjecture that Y_4 is a spanner for arbitrary point sets.

2 YY_4 is not a Spanner

We prove that YY_4 is not a spanner with the help of a simple counterexample. For a fixed small angular value $\theta > 0$, let a_0b_0 be an arbitrary line segment with slope θ. Let $b_0, b_1, \ldots, b_k$ be points equally distributed along a line of slope $\pi/2-\theta$, at intervals equal to $|a_0b_0|$ (so $|b_ib_{i+1}| = |a_0b_0|$, for each $i = 1, \ldots, k - 1$). See Fig. 1. Let $a_0, a_1, \ldots, a_k$ be points equally distributed along a line of slope $\pi/2 + \theta$, such that a_ib_i and a_jb_j are parallel, for each $0 \leq i, j \leq k$. Note that, for fixed

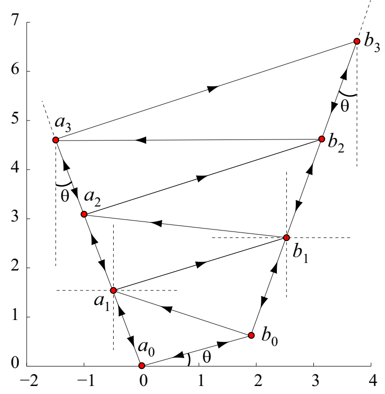


Figure 1: The Yao graph H induced by the point set V.

θ, a_0 and b_0, the point set $V = \{a_0, a_1, \ldots, a_k\} \cup \{b_0, b_1, \ldots, b_k\}$, is uniquely defined. For a fixed k, we select the angle θ small enough so that two lines including a_0a_1 and b_0b_1 are almost vertical.

Now note that in the Yao step, each node a_i selects the edges $\overrightarrow{a_ib_i}$, $\overrightarrow{a_ia_{i+1}}$ (if $i < k$) and $\overrightarrow{a_ia_{i-1}}$ (if $i > 0$). Similarly, each node b_i selects $\overrightarrow{b_ib_{i+1}}$ and $\overrightarrow{b_ia_{i+1}}$ (if $i < k$), $\overrightarrow{b_ib_{i-1}}$ (if $i > 0$), and $\overrightarrow{b_ia_i}$ (if $i = 0$). The result is the Yao graph H shown in Fig. 1.

In the reverse Yao step, since edges $\overrightarrow{a_ia_{i+1}}$ and $\overrightarrow{b_ia_{i+1}}$ lie in the same quadrant for a_{i+1} (for $i < k$),

[*]The first two authors have been supported by NSF grant CCF-0728909.

[†]Dept. Comput. Sci., Villanova Univ., Villanova, PA, USA. mirela.damian@villanova.edu.

[‡]Dept. Comput. Sci., Villanova Univ., Villanova, PA, USA. nawar.molla@villanova.edu.

[§]Dept. Math., Southern Connecticut State University, CT, USA. pinciuv1@southernct.edu.

and since $|a_i a_{i+1}| < |b_i a_{i+1}|$, the node a_{i+1} will eliminate $\overrightarrow{b_i a_{i+1}}$ from H. Similarly, since $\overrightarrow{a_i b_i}$ and $\overrightarrow{b_{i-1} b_i}$ lie in the same quadrant for b_i, and since $|b_{i-1} b_i| < |a_i b_i|$, the node b_i will eliminate $\overrightarrow{a_i b_i}$ from H. The result is the YaoYao graph H illustrated in Fig. 2. As $\theta \to 0$,

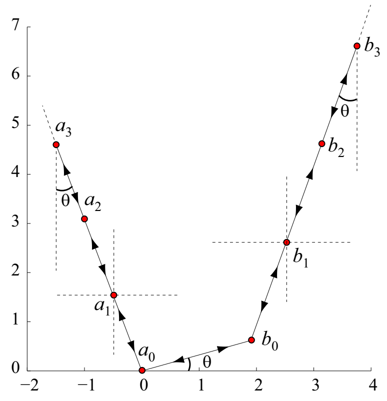


Figure 2: The graph H is not a spanner:

we have that $|a_k b_k| \to |a_0 b_0|$. However, a shortest path between a_k and b_k in the YaoYao graph H has length

$$\begin{aligned}\delta_H(a_k, b_k) &= \sum_{i=1}^{k} |a_i a_{i-1}| + |a_0 b_0| + \sum_{i=0}^{k-1} |b_i b_{i+1}| \\ &> (k+1) \cdot |a_0 b_0| = \Omega(n) \cdot |a_k b_k|\end{aligned}$$

Here $n = |V| = 2(k+1)$. The inequality above shows that there is no constant c such that $\delta_H(a_k, b_k) \le c \cdot |a_k b_k|$. It follows that H is not a spanner for the graph induced by the point set V.

3 Y_4 is a Spanner for Convex Position

We conjecture that Y_4 is a spanner for arbitrary point sets, however we only prove this conjecture for sets of points in convex position.

Let $u, v \in V$ be arbitrary. Let $Q(u, v)$ denote the cone (quadrant) with origin at u that contains v, and let $SQ(u, v)$ denote a smallest square centered at v whose boundary contains u. Note that $Q(u, v)$ and $SQ(u, v)$ are uniquely defined for a given pair of vertices u and v (see Fig. 3a). Let $R(u, v)$ denote the rectangle with diagonal uv.

For a fixed pair of vertices $u, v \in V$, we define recursively a directed path $\mathcal{P}(u \rightsquigarrow v)$ from u to v in Y_4 as follows. Let $u_0 = u$, and let $\overrightarrow{u_i u_{i+1}} \in Y_4$ be the edge that lies in $Q(u_i, v)$, for each i. Then

$$\mathcal{P}(u_i \rightsquigarrow v) = \begin{cases} \perp, & \text{if } u_i = v \\ \overrightarrow{u_i u_{i+1}} \oplus \mathcal{P}(u_{i+1} \rightsquigarrow v), & \text{otherwise.} \end{cases} \tag{1}$$

Here $\perp$ represents the null path and $\oplus$ represents the concatenation operator. This definition is illustrated in Fig. 3a. Fischer et al. [1] show the following properties of $\mathcal{P}(u \rightsquigarrow v)$:

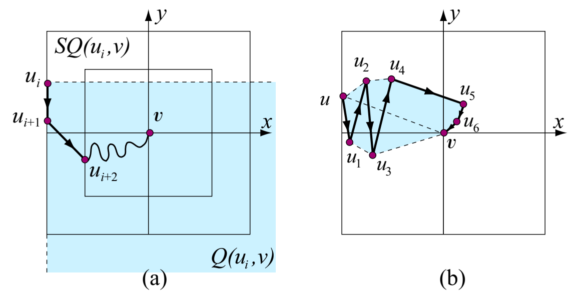


Figure 3: Constructing $\mathcal{P}(u \rightsquigarrow v)$. (a) The outer square is $SQ(u_i, v) = SQ(u_{i+1}, v)$, and the inner square is $SQ(u_{i+2}, v)$; shaded area is $Q(u_i, v)$; (b) $\mathcal{P}(u \rightsquigarrow v)$ lies inside $SQ(u, v)$.

Property 1 *Let $\mathcal{P}(u \rightsquigarrow v) = u_0, u_1, \ldots$ be as defined in (1). Then, for each $i \ge 0$,*

(a) $SQ(u_{i+1}, v) \subseteq SQ(u_i, v)$.

(b) $SQ(u_{i+2}, v) \subset SQ(u_i, v)$.

(c) $\mathcal{P}(u_i \rightsquigarrow v)$ lies entirely inside $SQ(u_i, v)$.

In the example from Fig. 3a for instance, $SQ(u_{i+1}, v) = SQ(u_i, v)$, however $SQ(u_{i+2}, v) \subset SQ(u_i, v)$. This latter property is critical to ensure that progress is made towards v and that the path $\mathcal{P}(u \rightsquigarrow v)$ is well defined. We now prove another important property of $\mathcal{P}(u \rightsquigarrow v)$.

Lemma 2 *$\mathcal{P}(u \rightsquigarrow v)$ does not self-intersect*[1].

Proof. The proof is by contradiction. Assume to the contrary that $P_{uv} = \mathcal{P}(u \rightsquigarrow v)$ is a self-intersecting path. Let $(u = u_0), u_1, u_2, \ldots$ be the vertices of P_{uv} in the order in which they appear from u to v along P_{uv}. Let $k > 0$ be the smallest index such $P_{uv}[u, u_k]$ does not self-intersect, but $P_{uv}[u, u_{k+1}]$ self-intersects (so $P_{uv}[u, u_{k+1}]$ contains at least three edges, meaning that $k > 1$). Then it must be that $u_k u_{k+1}$ intersects one of the edges on the subpath $P_{uv}[u, u_k]$. Let $i < k$ be such that $u_i u_{i+1}$ intersects $u_k u_{k+1}$ (see Fig. 4). Since $u_i u_{i+1}$ and $u_k u_{k+1}$ are non-adjacent, it must be that $k > i + 1$. This along with Property 1(b) implies that $SQ(u_{k+1}, v) \subset SQ(u_{i+1}, v)$. By Property 1(a), $SQ(u_{i+1}, v) \subseteq SQ(u_i, v)$. These together show that u_{k+1} cannot coincide with u_i or u_{i+1}, and therefore $u_k u_{k+1}$ and $u_i u_{i+1}$ do not share an endpoint. Thus, if $u_i u_{i+1}$ and $u_k u_{k+1}$ intersect, they must share a point interior to each of them. This along with the fact that u_{k+1} lies strictly inside $SQ(u_{i+1}, v)$ implies that $u_{k+1} \in R(u_i, u_{i+1})$. This in turn implies that $u_{k+1} \in$

[1]A path *self-intersects* if two non-adjacent edges share a point.

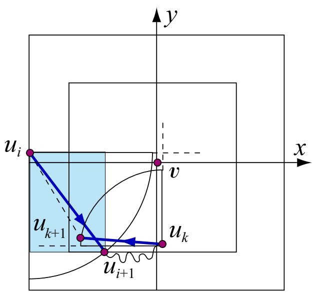


Figure 4: Edges $u_iu_j, u_ku_\ell \in \mathcal{P}(u \rightsquigarrow v)$ can not cross.

$Q(u_i, v)$ and $|u_iu_{k+1}| < |u_iu_{i+1}|$, contradicting the fact that u_iu_{i+1} is a shortest edge in $Q(u_i, v)$ (i.e, a Yao edge). This completes the proof. □

An immediate consequence of Lem. 2 is the following:

Corollary 3 *Let $\mathcal{P}(u \rightsquigarrow v) = (u = u_0), u_1, \ldots \ldots, u_{k-1}, (u_k = v)$. Then for any j, $0 \le j < k$, u_ju_{j+1} is on the convex hull of $u_j, u_{j+1}, \ldots, u_{k-1}, u_k$.*

The path $\mathcal{P}(u \rightsquigarrow v)$ can have an interesting oscillating behavior, as shown in Fig. 5. This is due to the fact

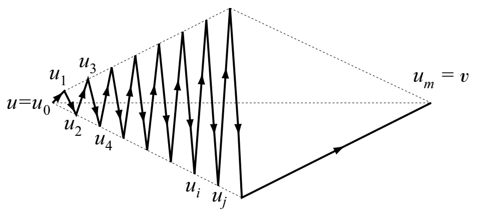


Figure 5: $\mathcal{P}(u \rightsquigarrow v)$ is arbitrarily long compared to $|uv|$.

that, at each intermediate point u_i, the path continues towards v along the Yao edge incident to u_i that lies in $Q(u_i, v)$, ignoring the incident (and potentially shorter) edges from adjacent quadrants. As a result, a vertex $u_j \in \mathcal{P}(u \rightsquigarrow v)$ that does not lie in $Q(u_i, v)$, but lies very close to u_i, might end up separated from u_i by a long subpath in $\mathcal{P}(u \rightsquigarrow v)$ (see $P_{uv}[u_i, u_j]$ from Fig. 5). We compensate for this shortcoming by refining $\mathcal{P}(u \rightsquigarrow v)$ in a second step, as described in the Y4Path algorithm below. The refinement step replaces longer subpaths of P_{uv} by shorter paths from Y_4, if possible.

For the example shown in Fig. 6 for instance, the Y4Path algorithm would replace the subpath $(\overrightarrow{u_2u_3}, \overrightarrow{u_3u_4}, \overrightarrow{u_4u_5})$ by $\mathcal{P}(u_5 \rightsquigarrow u_2)$. We will show that $\mathcal{P}(u_5 \rightsquigarrow u_2)$ has a nice behavior and does not interfere with the existing path P_{uv}. In particular, $\mathcal{P}(u_5 \rightsquigarrow u_2)$ lies exterior to $\mathcal{H}(P_{uv})$, in the region shaded in Fig. 6. This property is shown formally in the following lemma.

```
Algorithm Y4Path($Y_4, u, v$)

1. Compute $P_{uv} = \mathcal{P}(u \rightsquigarrow v)$.
   Let $\mathcal{H}(P_{uv})$ be the convex hull of $P_{uv}$.
   Let $\partial\mathcal{H}(P_{uv})$ be the boundary of $\mathcal{H}(P_{uv})$.

2. For each edge $u_iu_j \in \partial\mathcal{H}(P_{uv}) \setminus P_{uv}$, with $j > i > 0$
       If $|u_iu_j| < |u_iu_{i+1}|/2$
           Recompute $\mathcal{P}(u_j \rightsquigarrow u_i)$ as in (1).
           Replace $P_{uv}[u_i, u_j]$ in $P_{uv}$ by $\mathcal{P}(u_j \rightsquigarrow u_i)$.

Output $P_{uv}$ (undirected).
```

Table 1: The Y4Path algorithm.

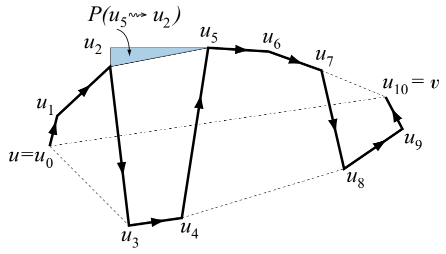


Figure 6: $\mathcal{P}(u_5 \rightsquigarrow u_2)$ lies in the shaded area.

Lemma 4 *Let u_iu_j be an edge processed in Step 2 of the* Y4Path *algorithm, with $i > 0$. Then $\mathcal{P}(u_j \rightsquigarrow u_i)$ lies in the closed rectangle $R = R(u_j, u_i)$.*

Proof. Let $Q_0 = Q(u_i, u_{i+1})$, and assume without loss of generality that $Q_1 = Q(u_i, u_j)$ lies counterclockwise from Q_0 (see Fig. 7). Let Q_2 and Q_3 be the other two quadrants with origin u_i, in counterclockwise order around u_i.

The proof is by contradiction. Let $\overrightarrow{w_kw_{k+1}} \in Q(w_k, u_i)$ be the first edge along $\mathcal{P}(u_j \rightsquigarrow u_i)$ that crosses the boundary of R. Thus $w_k \in R$ and $w_{k+1} \notin R$. Since $w_k \in R$, we have that

$$|w_ku_i| < |u_ju_i| \tag{2}$$

Since $w_{k+1} \in Q(w_k, u_i)$ and $\overrightarrow{w_kw_{k+1}}$ is a Yao edge, it must be that

$$|w_kw_{k+1}| \le |w_ku_i| \tag{3}$$

This implies that $w_{k+1} \notin Q_3$. Substituting (3) in the triangle inequality $|u_iw_{k+1}| < |u_iw_k| + |w_kw_{k+1}|$ yields $|u_iw_{k+1}| < 2|w_ku_i|$. Substituting (2) in this latter inequality yields

$$|u_iw_{k+1}| < 2|u_ju_i| \tag{4}$$

Recall that Step 2 of the Y4Path algorithm processes edges u_iu_j of length $|u_iu_j| < |u_iu_{i+1}|/2$. This together with (4) yields

$$|u_iw_{k+1}| < |u_iu_{i+1}| \tag{5}$$

Inequality (5) along with the fact that $\overrightarrow{u_i u_{i+1}}$ is a Yao edge, implies that $w_{k+1} \notin Q_0$. It also holds that $w_{k+1} \notin Q_1$, since $Q_1 \cap Q(w_k, u_i) \subseteq R$ and $w_{k+1} \in Q(w_k, u_i) \setminus R$.

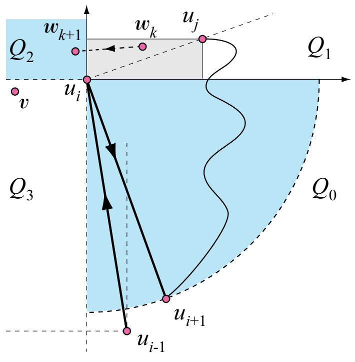


Figure 7: If $u_{i-1} \in Q_0$, then $u_{i+1} \notin Q(u_i, v)$.

It remains to discuss the case $w_{k+1} \in Q_2$ (see Fig. 7). Recall that $u_i u_j \in \partial\mathcal{H}(P_{uv})$. Also, by Cor. 3, $u_{i-1}u_i$ is a hull edge of $P_{uv}[u_{i-1}, v]$, so u_{i+1} is in the interior of the convex angle $\angle u_j u_i u_{i-1}$. Since $u_j \in Q_1$ and $u_{i+1} \in Q_0$, then either $u_{i-1} \in Q_3$ or $u_{i-1} \in Q_0$. If $u_{i-1} \in Q_3$, the quadrilateral $u_{i-1}u_{i+1}u_j w_{k+1}$ would contain u_i in its interior, contradicting the fact that V is in convex position. If $u_{i-1} \in Q_0$, since $\overrightarrow{u_{i-1}u_i}$ is a Yao edge, $v \in Q(u_{i-1}, u_i)$. Since $\overrightarrow{u_i u_{i+1}}$ is a Yao edge, $v \in Q(u_i, u_{i+1})$. It follows that $v \in Q(u_{i-1}, u_i) \bigcap Q(u_i, u_{i+1}) = R(u_{i-1}, u_i)$, contradicting the fact that $\overrightarrow{u_{i-1}u_i}$ is a Yao edge. Having exhausted all cases, we conclude that $w_k w_{k+1}$ cannot cross the boundary of R. □

The following corollary follows immediately from Lem. 4 and the fact that V is in convex position.

Corollary 5 *Let $u_i u_j$ be an edge processed in Step 2 of the* Y4PATH *algorithm, with $i > 0$. Then $\mathcal{P}(u_i \rightsquigarrow u_j) \subset \partial\mathcal{H}(P_{uv})$.*

Lemma 6 *If P is a convex polygon containing u and v that lies inside $SQ(u, v)$, then the perimeter of P is no more than $2(2 + \sqrt{2}) \cdot |uv|$.*

Proof. Let ab be a side of P containing v. Extend ab beyond each endpoint until it meets the boundary of $SQ(u, v)$ at two points, call them w_1 and w_2 (see Fig. 8). Then $w_1 w_2$ splits $SQ(u, v)$ into two congruent trapezoids (when $w_1 w_2$ is a diagonal of $SQ(u, v)$, the two trapezoids degenerate into two right triangles). Since P is convex, all points of P are on the same side of $w_1 w_2$, so P has to lie inside one of these trapezoids – call it T. The fact that $P \subseteq T$ along with the convexity property of P implies that the perimeter of P does not exceed the perimeter of T.

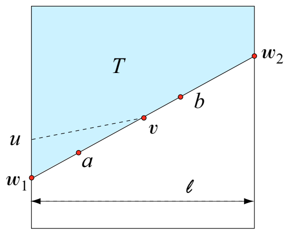


Figure 8: The perimeter of T does not exceed $(2 + \sqrt{2})\ell$.

Next we calculate the perimeter of T. Let ℓ be the side length of $SQ(u, v)$. Since w_1 and w_2 are symmetric with respect to v, the lengths of the two bases of T sum up to ℓ. The third side of T other than $w_1 w_2$ also has length ℓ. So the perimeter of T is

$$2\ell + |w_1 w_2| \le (2 + \sqrt{2})\ell$$

Since u is on the boundary of $SQ(u, v)$, $|uv| \ge \ell/2$. This along with the inequality above shows that the perimeter of P is no more than $2(2 + \sqrt{2})|uv|$. □

Theorem 7 *For any set V of points in convex position, the Yao graph Y_4 on V is a t-spanner, for $t = 4(2 + \sqrt{2})$.*

Proof. Cor. 5 implies that the only edges on P_{uv} crossing the interior of $\mathcal{H}(P_{uv})$ are those edges not processed in Step 2 of the Y4PATH algorithm. These are diagonals $u_i u_{i+1}$ of $\mathcal{H}(P_{uv})$ adjacent to $u_i u_j \in \partial\mathcal{H}(P_{uv}) \setminus P_{uv}$, with the property that $|u_i u_{i+1}| \le 2|u_i u_j|$. It follows that P_{uv} is no longer than twice the perimeter of $\partial\mathcal{H}(P_{uv})$. This along with Lem. 6 concludes the proof. □

Concluding Remarks. We have shown that YY_4 is not a spanner, and that Y_4 is a spanner for points in convex position. Proving that Y_4 is a spanner for arbitrary point sets remains open.

References

[1] M. Fischer, T. Lukovszki, and M. Ziegler. Geometric searching in walkthrough animations with weak spanners in real time. In *ESA '98: Proc. of the 6th Annual European Symposium on Algorithms*, pages 163–174, 1998.

[2] X. Li, P. Wan, Y. Wang, and O. Frieder. Sparse power efficient topology for wireless networks. In *HICSS'02: Proc. of the 35th Annual Hawaii Int. Conference on System Sciences*, volume 9, page 296.2, 2002.

[3] A.C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Wiener Index and Diameter of a Planar Graph in Subquadratic Time

Christian Wulff-Nilsen*

Abstract

We solve two open problems by proving the existence of subquadratic time algorithms for computing the Wiener index, defined as the sum of all-pairs shortest path distances, and the diameter, defined as the maximum distance between any vertex pair, of an unweighted planar graph. We do this by exhibiting algorithms with $O(n^2 \log\log n/\log n)$ running time and $O(n)$ space requirement where n is the number of vertices of the graph.

1 Introduction

A *molecular topological index* is a value obtained from the graph structure of a molecule such that this value (hopefully) correlates with physical and/or chemical properties of the molecule. Perhaps the most studied molecular topological index is the so called *Wiener index*, a generalization of a definition given by Wiener in 1947 [8]. The Wiener index of a graph (weighted or unweighted) is defined as the sum of distances between all pairs of vertices of the graph. Computing the Wiener index is essentially equivalent to computing the average distance between vertex pairs.

The Wiener index can clearly be computed in the amount of time it takes to compute APSP distances. For special types of graphs, faster algorithms are known. Linear time algorithms are known for cactii [9] and benzenoid systems [2] and recently Cabello and Knauer [1] gave near-linear time algorithms for graphs of bounded treewidth. More specifically, they showed that the Wiener index of an n-vertex graph of treewidth $k \geq 3$ can be found in $O(n \log^{k-1} n)$ time. All bounds hold for graphs with arbitrary non-negative edge weights.

For planar graphs, the Wiener index can be found in quadratic time using the algorithm of Frederickson [4]. One of the main open problems in this context concerns the existence of a subquadratic time algorithm for such graphs.

Another important quantity is the *diameter* of a graph, defined as the maximum distance between any vertex pair. With Frederickson's algorithm, the diameter of a planar graph can be found in quadratic time but it is open whether a subquadratic time algorithm exists (Problem 6.2 in [3]).

*Department of Computer Science, University of Copenhagen, koolooz@diku.dk

In this paper, we solve both of these open problems for planar unweighted graphs. We do this by exhibiting algorithms with running time $O(n^2 \log\log n/\log n)$ and space requirement $O(n)$ where n is the number of vertices.

The organization of the paper is as follows. In Section 2, we give various definitions and introduce some notation. We mention a result by Frederickson [4] in Section 3, a result that allows us to divide a planar graph into regions with some nice properties. In Section 4, we rely on this result to obtain our subquadratic time algorithm for computing the Wiener index of a planar graph. In Section 5, we show how a simple modification gives an algorithm with the same time bound for computing the diameter of a planar graph. Finally, we make some concluding remarks in Section 6.

2 Definitions and Notation

In all the following, $G = (V, E)$ is an unweighted planar graph with n vertices. For $u, v \in V$, we let $d_G(u, v)$ denote the length of a shortest path in G between u and v.

Given a subgraph H of G, we let V_H denote its vertex set.

Given subsets $U_1, U_2 \subseteq V$, we define

$$\sum(U_1, U_2) = \sum_{u \in U_1} \sum_{v \in U_2} d_G(u, v).$$

We omit G in the notation but this should not cause any confusion. For a vertex u and a subset U of V, we write $\sum(U, u)$ as a shorthand for $\sum(U, \{u\})$.

We let $\sum G$ denote the sum of all-pairs shortest path distances in G, i.e. $\sum G = \frac{1}{2}\sum(V, V)$, and we refer to it as the *Wiener index* of G.

A *region* of G is a subset R of vertices of G. A *boundary vertex* of R is a vertex in R which is adjacent in G to a vertex in $V \setminus R$. Vertices of R that are not boundary vertices are called *interior vertices* (of R).

We let log denote the base 2 logarithm.

3 r-division of a Planar Graph

By applying the separator theorem of Lipton and Tarjan [6] recursively to a given planar graph, Frederickson [4] obtained the following result which we state as a lemma.

Lemma 1 *Given a parameter $r \in (0, n)$ (which may depend on n), an n-vertex planar graph can be divided into $\Theta(n/r)$ regions each of which contains at most r vertices and $O(\sqrt{r})$ boundary vertices. Furthermore, each interior vertex is contained in exactly one region. Finding such a division can be done in $O(n \log n)$ time.*

For parameter r, we refer to the division in Lemma 1 as an *r-division* (of the graph). If $R_1, \ldots, R_k \subseteq V$ are the regions obtained, we denote the r-division by the tuple $(R_1, \ldots, R_k)$.

Finding an r-division for a suitable value of r is essential in our algorithms.

4 Wiener Index of a Planar Graph

In this section, we show how to compute the Wiener index $\sum G$ of G in $O(n^2 \log\log n/\log n)$ time. We will assume that G is connected since otherwise the problem is trivial.

The first step of our algorithm is to compute an r-division $(R_1, \ldots, R_k)$ of G for some parameter r which we specify later. For now, just regard r as some function of n. We will show how to compute $\sum G$ in $O(n^2/\sqrt{r} + nr^{O(\sqrt{r})})$ time. From this and from a suitable choice of r, the first result of the paper will follow.

In the following, let B be the set of boundary vertices over all regions $R_1, \ldots, R_k$. We precompute shortest path distances from each vertex in B to all vertices in V. Since $|B| = O(n/\sqrt{r})$, this can be done in $O(n^2/\sqrt{r})$ time using the linear time SSSP algorithm in [5] for each vertex in B. From these distances, we obtain values $\sum(B, V)$ and $\sum(B, B)$ in $O(n^2/\sqrt{r})$ time.

Observe that $\sum(B, V) - \frac{1}{2}\sum(B, B)$ is the sum of all shortest path distances in G between vertex pairs (u, v) for which either u or v (or both) is a boundary vertex. Since, by Lemma 1, each interior vertex belongs to exactly one region, we can thus obtain $\sum G$ as the sum

$$\sum(B, V) - \frac{1}{2}\sum(B, B) +$$
$$\frac{1}{2}\sum_{i=1}^{k}\sum(R_i \setminus B, V \setminus (R_i \cup B)) + \sum(R_i \setminus B, R_i \setminus B).$$

Let R be one of the regions $R_1, \ldots, R_k$. In the following, we focus on the problem of computing $\sum(R \setminus B, V \setminus (R \cup B))$ and $\sum(R \setminus B, R \setminus B)$. If we can show that these two quantities can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time, it will follow that $\sum G$ can be computed in $O(n^2/\sqrt{r} + nr^{O(\sqrt{r})})$ time since $k = \Theta(n/r)$.

Let us start with the easy part, that of computing $\sum(R \setminus B, R \setminus B)$. We do this by computing shortest path distances in G between each pair of vertices in R. To do this efficiently, we take the subgraph of G induced by R and add to it an edge between each pair of boundary vertices of R; the length of this edge is equal to the distance in G between those two vertices (we do not add an edge between a pair of boundary vertices already connected by an edge). We then run an APSP algorithm like Floyd-Warshall on the resulting graph.

Since shortest path distances from boundary vertices of R to all vertices in G (and in particular to all boundary vertices in R) have been precomputed, it follows that we can compute shortest path distances in G between each pair of vertices in R in $O(r^3)$ time. Hence, we can compute $\sum(R \backslash B, R \backslash B)$ in $O(r^3)$ time.

Now, to compute $\sum(R \setminus B, V \setminus (R \cup B))$, let $C_1, \ldots, C_s$ be the connected components of the subgraph of G induced by R. Then

$$\sum(R \setminus B, V \setminus (R \cup B)) =$$
$$\sum_{i=1}^{s}\sum(V_{C_i} \setminus B, V \setminus (R \cup B)). \qquad (1)$$

Let C be one of these connected components, let $n_C = |V_C|$, and let $p_1, \ldots, p_t$ be the boundary vertices of R belonging to C. In the following, we show how to compute $\sum(V_C \setminus B, V \setminus (R \cup B))$ in $O(nt + n_C^{O(t)})$ time. It will then follow that the left-hand side of (1) can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time since each of the $O(\sqrt{r})$ boundary vertices of R belongs to exactly one connected component.

To compute $\sum(V_C \setminus B, V \setminus (R \cup B))$, the basic idea is the following. Given some vertex $u \in V \setminus (R \cup B)$, suppose we have precomputed, for each boundary vertex p_i, $i = 1, \ldots, t$,

1. the number $n_{i,u}$ of vertices v in $V_C \setminus B$ for which i is the smallest j such that $d_G(u, v) = d_G(u, p_j) + d_C(p_j, v)$,
2. the sum $D_{i,u}$ of distances in C from p_i to each of these vertices in $V_C \setminus B$.

Then

$$\sum(V_C \setminus B, u) = \sum_{i=1}^{t} n_{i,u} d_G(u, p_i) + D_{i,u}, \qquad (2)$$

see Figure 1.

Given these precomputations, we can thus obtain $\sum(V_C \setminus B, u)$ in $O(t)$ time and from this it follows that $\sum(V_C \setminus B, V \setminus (R \cup B))$ can be computed in $O(nt)$ time.

In order to perform the above precomputations efficiently we need the following key observation.

Lemma 2 *Let $u \in V \setminus (R \cup B)$ and let $i \in \{1, \ldots, t\}$ be given. Then $n_{i,u}$ and $D_{i,u}$ are completely determined by shortest path distances in C and values $d_G(u, p_j) - d_G(u, p_1)$ for $j = 1, \ldots, t$.*

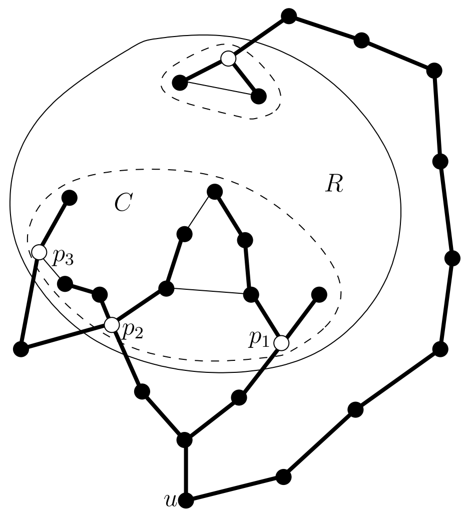


Figure 1: Graph instance for which component C has $t = 3$ boundary vertices p_1, p_2, and p_3 of R and where $(n_{1,u}, D_{1,u}) = (4, 7)$, $(n_{2,u}, D_{2,u}) = (4, 6)$, and $(n_{3,u}, D_{3,u}) = (1, 1)$. Also $\sum(V_C \setminus B, u) = \sum_{i=1}^{t} n_{i,u} d_G(u, p_i) + D_{i,u} = 43$.

Proof. Let $v \in V_C \setminus B$. Any path from u to v in G must contain at least one of the boundary vertices $p_1, \ldots, p_t$. Hence, the following two conditions are equivalent:

1. i is the smallest j such that $d_G(u, v) = d_G(u, p_j) + d_C(p_j, v)$,
2. $d_G(u, p_i) - d_G(u, p_j) \leq d_C(v, p_j) - d_C(v, p_i)$ holds for $1 \leq j \leq t$ with strict inequality for $1 \leq j < i$.

Since $d_G(u, p_i) - d_G(u, p_j) = (d_G(u, p_i) - d_G(u, p_1)) - (d_G(u, p_j) - d_G(u, p_1))$, the above shows that $n_{i,u}$ depends only on shortest path distances in C and values $d_G(u, p_j) - d_G(u, p_1)$, $j = 1, \ldots, t$. Clearly, this also holds for $D_{i,u}$. $\square$

Before proceeding, let us define a map $\phi : V \setminus (R \cup B) \to \mathbb{Z}^t$ by

$$\phi(u)[j] = d_G(u, p_j) - d_G(u, p_1),$$

for $j = 1, \ldots, t$. Let p be a point in $\phi(V \setminus (R \cup B))$ and let u be a vertex in $V \setminus (R \cup B)$ such that $\phi(u) = p$. Associate with p values $n_p(i)$ and $D_p(i)$ for $i = 1, \ldots, t$, defined by

$$n_p(i) = n_{i,u},$$
$$D_p(i) = D_{i,u}.$$

By Lemma 2, this is well-defined since $n_p(i)$ and $D_p(i)$ do not depend on the choice of $u \in \phi^{-1}(\{p\})$.

The strategy now is to precompute n_p- and D_p-values for each $p \in \phi(V \setminus (R \cup B))$ and then, for each $u \in V \setminus (R \cup B)$, compute $p = \phi(u)$ and obtain, for $i = 1, \ldots, t$, $n_{u,i}$ and $D_{u,i}$ as the precomputed values $n_p(i)$ and $D_p(i)$, respectively. Function ϕ will act in a way similar to a hash function in that it maps a key (a vertex u) into a hash (the point $p = \phi(u)$) to obtain a value ($n_p(i)$ and $D_p(i)$ for $i = 1, \ldots, t$).

For this strategy to work well, we need the following lemma which shows that the number of points in $\phi(V \setminus (R \cup B))$ is small compared to $V \setminus (R \cup B)$ and hence that we only need to compute a small number of n_p- and D_p-values.

Lemma 3 $\phi(V \setminus (R \cup B)) \subseteq \{-(n_C - 1), \ldots, n_C - 1\}^t$.

Proof. Let $u \in V \setminus (R \cup B)$ and let $j \in \{1, \ldots, t\}$ be given. Since C is connected there is a simple path in C from p_1 to p_j and this path consists of at most $n_C - 1$ edges. Thus, $d_G(p_1, p_j) \leq d_C(p_1, p_j) \leq n_C - 1$ and the triangle inequality implies

$$-(n_C - 1) \leq d_G(u, p_j) - d_G(u, p_1) \leq n_C - 1.$$

This shows the lemma since $\phi(u)[j] = d_G(u, p_j) - d_G(u, p_1)$. $\square$

We are now ready to describe how to compute $\sum(V_C \setminus B, V \setminus (R \cup B))$. We first initialize a t-dimensional table T with an entry $T[p]$ for each point $p \in \{-(n_C - 1), \ldots, n_C - 1\}^t$. Associated with $T[p]$ are two t-dimensional vectors to hold values $(n_p(1), \ldots, n_p(t))$ and $(D_p(1), \ldots, D_p(t))$. Initially, all entries of T are unmarked. The initialization step takes a total of $O(t(2n_C - 1)^t) = O(n_C^{O(t)})$ time.

For each $u \in V \setminus (R \cup B)$, we compute point $p = \phi(u)$ in $O(t)$ time (this is possible since SSSP distances in G have been precomputed for each boundary vertex). Assume first that entry $T[p]$ is unmarked. Then we mark it and compute the n_p- and D_p-values and store them in the vectors associated with $T[p]$. By Lemma 2, computing and storing these values can clearly be done in time polynomial in n_C (a weak analysis but it suffices). From these values, we compute $\sum(V_C \setminus B, u)$ in $O(t)$ time.

If $T[p]$ is already marked then we do not compute n_p- and D_p-values. Instead we perform a lookup in T at entry $T[p]$ to obtain $\sum(V_C \setminus B, u)$ in $O(t)$ time.

Clearly, we compute n_p- and D_p-values at most once for each $p \in \{-(n_C - 1), \ldots, n_C - 1\}^t$. It follows that the above algorithm computes $\sum(V_C \setminus B, V \setminus (R \cup B))$ in $O(nt + n_C^{O(t)})$ time.

From the above and from (1), we get the following result.

Lemma 4 *For each region R, values $\sum(R \setminus B, V \setminus (R \cup B))$ and $\sum(R \setminus B, R \setminus B)$ can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time assuming shortest path distances from each boundary vertex of R to each vertex in G have been precomputed.*

We are now ready for our first result.

Theorem 5 *The Wiener index $\sum G$ of an unweighted planar n-vertex graph G can be computed in $O(n^2 \log\log n/\log n)$ time and $O(n)$ space.*

Proof. Computing shortest path distances from each boundary vertex to each vertex in G can be done in $O(n^2/\sqrt{r})$ time.

Applying Lemma 4 to each of the $\Theta(n/r)$ regions in the r-division of G gives us $\sum G$ in $O(n^2/\sqrt{r}+nr^{c'\sqrt{r}})$ time for some constant c'.

We pick $r = (c\log n/\log\log n)^2$ where $c > 0$ is some constant (to be specified). For $n > 2^c$ we have $\log n > c$, and for $n > 4$ we have $\log\log n > 1$. Thus, for $n > \max\{2^c, 4\}$,

$$\begin{aligned} r^{c'\sqrt{r}} &< (c\log n)^{2c'c\log n/\log\log n} \\ &< (\log n)^{4c'c\log n/\log\log n} \\ &= n^{4c'c}, \end{aligned}$$

since $(\log n)^{\log n/\log\log n} = n$. It follows that if we choose $c < 1/(4c')$, we have $r^{c'\sqrt{r}} = O(n^\epsilon)$, where $\epsilon < 1$. With this choice of r, the total running time of the algorithm is

$$\begin{aligned} O(n^2/\sqrt{r} + nr^{c'\sqrt{r}}) &= O(n^2\log\log n/\log n + n^{1+\epsilon}) \\ &= O(n^2\log\log n/\log n), \end{aligned}$$

as requested.

Simple modifications of the algorithm described above allows us to obtain linear space requirement without affecting running time. Due to space constraints, we omit the details. □

5 Diameter of a Planar Graph

The following theorem shows that we can obtain a similar time bound for computing the diameter of an unweighted planar graph.

Theorem 6 *The diameter of an n-vertex planar graph with non-negative edge weights can be computed in $O(n^2\log\log n/\log n)$ time and $O(n)$ space.*

Proof. We can apply ideas similar to those above for the Wiener index problem. The only essential difference is that instead of variables $n_{i,u}$ and $D_{i,u}$ (see Section 4) we introduce variables $L_{i,u}$ for each i and u that keep track of the longest distance in C from p_i to the set of vertices specified in the definition of $n_{i,u}$. We can then use the identity

$$\max(V_C \setminus B, u) = \max_{1\le i\le t}\{d_G(u,p_i) + L_{i,u}\}$$

instead of (2). □

Note that our two algorithms do not rely on planarity except when computing an r-division and SSSP distances in linear time. Our results can therefore be extended to the larger class of unweighted subgraph-closed $\sqrt{n}$-separable graphs for which separators can be found efficiently [5].

6 Conclusion

We solved two open problems, the existence of subquadratic time algorithms for computing the Wiener index and diameter of an unweighted planar graph. We did this by exhibiting $O(n^2\log\log n/\log n)$ time algorithms where n is the number of vertices. Both algorithms have linear space requirement.

It remains open whether "truly" subquadratic time algorithms exist, i.e. algorithms with $O(n^c)$ running time for constant $c < 2$.

In a forthcoming paper, we extend our results to planar graphs with arbitrary non-negative edge weights. The new ideas we introduce in that paper allow us to solve also the stretch factor problem for planar graphs in subquadratic time. In another paper, we show how similar ideas give a faster algorithm for computing shortest path distances between $k = O(n^2)$ pairs of vertices in a planar graph for large k.

References

[1] S. Cabello and C. Knauer. *Algorithms for graphs of bounded treewidth via orthogonal range searching.* Manuscript, Berlin, 2007.

[2] V. Chepoi and S. Klavžar. *The Wiener index and the Szeged index of benzenoid systems in linear time.* J. Chem. Inf. Comput. Sci., 37:752–755, 1997.

[3] F. R. K. Chung. *Diameters of Graphs: Old Problems and New Results.* Congressus Numerantium, 60:295–317, 1987.

[4] G. N. Frederickson. *Fast algorithms for shortest paths in planar graphs, with applications.* SIAM J. Comput., 16 (1987), pp. 1004–1022.

[5] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. *Faster Shortest-Path Algorithms for Planar Graphs.* Journal of Computer and System Sciences volume 55, issue 1, August 1997, pages 3–23.

[6] R. J. Lipton and R. E. Tarjan. *A Separator Theorem for Planar Graphs.* STAN-CS-77-627, October 1977.

[7] B. Mohar and T. Pisanski. *How to compute the Wiener index of a graph.* J. Math. Chem., pages 267–277, 1988.

[8] H. Wiener. *Structural determination of paraffin boiling points.* J. Amer. Chem. Sot., 69:17–20, 1947.

[9] B. Zmazek and J. Žerovnik. *Computing the weighted Wiener and Szeged number on weighted cactus graphs in linear time.* Croatica Chemica Acta, 76:137–143, 2003.

Tailored Bregman Ball Trees for Effective Nearest Neighbors[*]

Frank Nielsen[†] Paolo Piro, Michel Barlaud[‡]

Abstract

Nearest Neighbor (NN) search is a crucial tool that remains critical in many challenging applications of computational geometry (e.g., surface reconstruction, clustering) and computer vision (e.g., image and information retrieval, classification, data mining). We present an effective Bregman ball tree [5] (Bb-tree) construction algorithm that *adapts* locally its internal node degrees to the inner geometric characteristics of the data-sets. Since symmetric measures are usually preferred for applications in content-based information retrieval, we furthermore extend the Bb-tree to the case of symmetrized Bregman divergences. Exact and approximate NN search experiments using high-dimensional real-world data-sets illustrate that our method improves significantly over the state of the art [5], sometimes by an *order* of magnitude.

1 Introduction and prior work

Finding nearest neighbors (NNs) is a very common task occurring in many applications ranging from computer vision to machine learning and data mining. Let $\mathcal{S} = \{p_1, ..., p_n\}$ be a set of n d-dimensional points (with d typically ranging up to a few thousand dimensions). Given a *query* point q, the nearest neighbor $\mathrm{NN}(q)$ is defined to be the "closest" point of $\mathcal{S}$ with respect to a dissimilarity measure D: $\mathrm{NN}(q) = \arg\min_i D(q, p_i)$. Instead of considering the closest neighbor, queries can be enlarged to report the first k "closest" points (useful for k-NN classification rule in machine learning). It is usually enough to get a *good* neighbor [6] by relaxing the exact search to get *near* and *fast* neighbors. Besides the theoretical puzzling questions related to the dreaded curse of dimensionality [6], practitioners make every endeavor to speed up applications by designing tailored schemes for improving over the naïve linear-time $O(dn)$ brute-force method. Among the flourishing literature of nearest neighbor techniques, we distinguish two main sets of methods: (1) those relying on tree-like space partitions with branch-and-bound queries, such as kD-trees, metric ball and vantage point trees [5, 8], and (2) those based on mapping techniques [3] (e.g., locality-sensitive hashing, random projections). The former tree-based methods improve over the brute force algorithm by *pruning* sub-trees whenever traversing the trees with queries (exact NN) or stopping after visiting a given "budget" of leaves (approximate NN). The latter methods concentrate on reducing dimensions while preserving as much as possible distances and controlling geometrically collisions of hash functions (only work for approximate NN). Since the Euclidean distance is often inappropriate for meaningfully measuring the proximity of feature points arising from, say, image applications, a wide range of distortion measures has been proposed and NN data-structures have been first extended to arbitrary metrics (e.g., vantage point trees [8]). However, these NN search methods rely fundamentally on the *triangle inequality* property, which is not satisfied by information-theoretic statistical distances such as the Kullback-Leibler divergence. Cayton [5] recently proposed the analogous of metric ball trees for the broad class of Bregman divergences. Bregman divergences D_F on vector set $\mathcal{X} \subset \mathbb{R}^d$ are defined for a strictly *convex* and *differentiable* generator $F(x) : \mathcal{X} \subset \mathbb{R}^d \mapsto \mathbb{R}$ as $D_F(p||q) = F(p) - F(q) - (p-q)^T \nabla F(q)$, where ∇F denotes the gradient of F. Those divergences (parameterized by a generator F) include all quadratic distances (also known as Mahalanobis squared distances, which are the only symmetric Bregman divergences and are obtained for $F(x) = \Sigma^{-1} x$, where $\Sigma \succ 0$ is the positive-definite variance-covariance matrix) and the asymmetric KL divergence ($F(x) = \sum_{j=1}^d x_j \log x_j$, the negative Shannon entropy), for which $D_F(p||q) \neq D_F(q||p)$. Many fundamental algorithms (e.g., k-means [4], PCA) and data-structures (e.g., Voronoi diagrams [10]) have been generalized to that class of divergences, thus offering *meta-algorithms* that can work for *any* Bregman divergence. For example, Banerjee et al. [4] showed that the celebrated Lloyd k-means algorithm extends to (and only to) the class of Bregman divergences unifying various algorithms. Since Bregman divergences are typically non-metric asymmetric distortion measures, we consider both left/right-sided and symmetrized NN searches defined as follows: $\mathrm{NN}^r_F(q) = \arg\min_i D_F(p_i||q)$ (right-sided), $\mathrm{NN}^l_F(q) = \arg\min_i D_F(q||p_i)$ (left-sided) and $\mathrm{NN}_F(q) = \arg\min_i (D_F(p_i||q) + D_F(q||p_i))/2$ (sym-

[*]Financially supported by ANR-07-BLAN-0328-01 GAIA (Computational Information Geometry and Applications) and DIGITEO GAS 2008-16D (Geometric Algorithms & Statistics)

[†]École Polytechnique, LIX, Palaiseau, France, nielsen@lix.polytechnique.fr

[‡]CNRS I3S, Sophia-Antipolis, France, {piro,barlaud}@i3s.unice.fr

metrized).

2 Bregman ball trees (Bb-trees)

Without loss of generality, we consider only *right-sided* NN queries, as left-sided NN queries can be handled similarly by considering the dual divergence $D_{F^*}(\nabla F(q)||\nabla F(p)) = D_F(p||q)$ arising from the Legendre-Fenchel conjugate F^* of F (with ∇F^* the functional inverse of ∇F^{-1}). See [10, 5] for details. Bregman generators come by pairs (F, F^*), e.g. the dual Legendre conjugates $F(x) = x \log x$ and $F^*(y) = \exp y$ (with $F'(x) = (F^{*\prime}(y))^{-1}$), see [10].

2.1 Outline of Bregman ball trees (Bb-trees) [5]

Similar to metric ball trees, a Bb-tree is built in a *top-down* fashion by applying recursively a partitioning scheme. First, the root is created to handle the source data-set $\mathcal{S}$. A 2-means (k-means [4] with $k = 2$) is computed, splitting $\mathcal{S}$ according to the two centroid points, say c_l and c_r, with respect to D_F. These two centroids defines a partition $\mathcal{S} = \mathcal{S}_l \cup \mathcal{S}_r$ that can be covered geometrically with corresponding Bregman balls $B(c_l, R_l)$ and $B(c_r, R_r)$ (possibly overlapping). This hierarchical decomposition of $\mathcal{S}$ is carried out recursively on $\mathcal{S}_l$ and $\mathcal{S}_r$ until a stopping criterion is eventually met. Two such typical termination criteria are: (1) a predefined maximum number of points l_0 (stored at leaves), or (2) a prescribed maximum radius (r_0, comes in handy for approximate NN). Note that the source points are stored *only* at leaves of the Bb-tree, and the sub-sets $\mathcal{S}_l$ and $\mathcal{S}_r$ may be theoretically unbalanced. Internal nodes store only two left/right Bregman balls covering the point sets stored at their left/right sub-trees.

In both *exact* and *approximate* NN queries, we perform a *branch-and-bound* search to answer queries. For a given query q, the tree is traversed in *depth-first-search* order from the root to the leaves. At an internal node, we choose to branch first on the sub-tree whose corresponding ball is "closer" to the query q (the sibling is temporarily ignored). Once a leaf node is reached, the closest point to q among the points stored at the leaf is computed using the brute-force method. This first visited leaf yields the very first NN point candidate p', thus giving an upper bound $D_F(p'||q)$ to the NN distance. In exact search, the tree traversal goes on through all formerly ignored subtrees. To decide whether a subtree must be explored or not, we check whether $D_F(p'||q) > \min_{x \in B(c,R)} D_F(x||q)$, where p' is the current NN candidate. This test is performed by projecting q onto the Bregman ball $B(c, R)$. The projection of a point q onto a Bregman ball B is the unique minimizer $q_B = \arg\min_{x \in B} D_F(x||q)$ [10]. Instead of projecting exactly the query point onto the ball, we rather make use of a *Bregman annulus* $B(c, R, R') = \{x \mid R \leq D_F(x||c) \leq R'\}$ that encloses the exact projection by construction: $q_B \in B(c, R, R')$. If $D_F(p'||q) < R$ then the node can be pruned; If $D_F(p'||q) > R'$ then the node must be explored. These lower/upper bounds are computed during the geodesic bisection search of the projection [5] (see Section 2.4). Overall, observe that the less the number of pairwise intersecting balls stored at nodes, the better the Bb-tree performance. Although exact NN retrieval on Bb-trees can often be achieved with much smaller computational cost than the brute-force search, the practical interest of Bb-trees is to get **significant** speed-up search when performing **approximate** NN queries. The approximate search allows one for large speed-up as it does not perform exhaustive branch-and-bound search. A common criterion to perform approximate search is to stop the branch-and-bound algorithm after exploring a *prescribed* number of leaves [5].

2.2 Speeding up construction time: `Bb-tree++`

In order to preprocess datasets efficiently by means of Bregman ball trees, instead of running the full regular Bregman k-means algorithm [4], we just perform a careful light initialization of the two cluster centers (seeds). Initialization turns out to be the crucial stage of k-means. That is, k-means locally optimizes the potential $P(\mathcal{C}) = \sum_{p \in \mathcal{S}} \min_{c \in \mathcal{C}} D_F(p||c)$, where $\mathcal{C}$ denotes the set of k centers. Each round assignment/cluster adjustment decreases this potential function so that monotonous convergence is guaranteed. It is striking to know that the worst-case running-time of k-means is theoretically exponential with the dimension although a recent smooth analysis yields polynomial amortized time [9]. A bad initialization may trap k-means into a local optimum. Initialization is thus all the more important, as k-means is called at each internal node of the Bb-tree to partition the data-sets into two sub-sets. Therefore, the idea is to replace the k-means local iterative algorithm by a well-chosen initialization that provides a guaranteed upper-bound on the optimal partition. This quantum leap for k-means was discovered by Arthur and Vassilvitskii [2] and later extended to Bregman divergences by Nock et al. [12]. We describe the initialization procedure of Bregman 2-`means++` for a set $\mathcal{S}$. First, draw uniformly at random the seed c_l (cluster "center") among the points of the data-set. Then compute the Bregman divergence of this point to all points of $\mathcal{S}$, and draw the second seed c_r according to the divergence distribution: $\pi_i = \frac{D_F(p_i||c_l)}{\sum_{p_j \in \mathcal{S}} D_F(p_j||c_l)}$. Thus c_r can never be c_l, since $D_F(c_l||c_l) = 0$ (the probability of drawing c_l is zero). This careful initialization yields an extremely fast tree construction, which statistically provides nice splitting and tends to balance sub-sets stored at leaves. Arthur and Vassilvitskii [2] proved that this initialization yields a k-means score $P(\mathcal{C})$ at most $8(2+\log k)$ of the optimal value. Similar bounds in $O(\log k)$ for generic Bregman divergences were later reported by Nock et al. [12]. Note that the two left/right Bregman balls stored at internal nodes tend to minimize the Bregman information [4] (i.e., variance for the square potential $F(x) = x^2$, mutual information for the negative Shannon entropy $F(x) = x \log x$, etc). Next, we improve the partition at each internal node by *learning* its degree (number of siblings) from the local data-sets.

2.3 Learning the tree branching factor

Answering queries may range from optimal logarithmic time (i.e., the shortest path to a leaf) up to linear time for a complete tree traversal. Therefore, it is important to partition the source data-sets into as many as possible non-overlapping Bregman balls. However, consider a data-set consisting of three separated Gaussian point samples. Forcing this set to be split into two will likely create overlapping balls. Thus we better learn the number of sub-sets when partitioning the data so that the induced Bregman balls better fit the intrinsic geometric characteristics of the set.[1] We adapt the branching factor bf_i (up to a maximum branching factor BF) of each internal node of the Bb-tree to the underlying distribution of the points by using the G-means strategy [7]. The underlying idea is to assume Gaussian distribution of each group of points (hence the name G-means).[2] Our use of G-means [7] algorithm starts by setting $k = 2$ and then test for Gaussian distribution of the points using the Anderson-Darling statistical test. Given a confidence level α, if the Anderson-Darling normality test returns `true`, we keep the center, otherwise we split it into two. Between two rounds, we simply run Bregman k-means++ initialization on the data-set and get all the new centers that hopefully refine the partitioning. We enforce a maximum degree to each internal node in order to strike a balance between the average tree depth and the overall ball tree shape. The initialization to k clusters follows the same principle: Namely, we draw the l-th seed from the data-set uniformly according to the distribution $\pi_i = \frac{D_F(p_i||\mathcal{C}_{l-1})}{\sum_{p_j \in \mathcal{S}} D_F(p_j||\mathcal{C}_{l-1})}$, where $\mathcal{C}_{l-1}$ denotes the formerly chosen $(l-1)$-th seed and $D_F(p||\mathcal{S}) = \min_{x \in \mathcal{S}} D_F(p||x)$.

When visiting the Bb-tree for answering nearest neighbor queries, we use a *priority queue*. To relax the exact NN to approximate NN queries, the criterion of stopping the search once a few leaves have been visited was proposed and successfully demonstrated by Cayton [5]. However, there was no guarantee to get a good approximation to the exact NN because leaves containing sub-sets that are close to the exact NN are not necessarily close in the tree. We improve this point by a careful **non-recursive** implementation of the tree traversal, which allows us to order the nodes to be explored by their divergence to the query point (i.e., the divergence of the query to the ball centers). Using a priority queue guarantees that nearest nodes are always explored first when traversing back the tree, by jumping appropriately to the most likely not yet visited sub-tree.

[1]Refer to Figure 1 of [7] for examples.

[2]This is not restrictive as any smooth density function may be arbitrarily well approximated using a Gaussian mixture model.

2.4 Bregman projection onto balls

The geodesic Γ_{pq} linking p to q is defined as $\Gamma_{pq} = \{\mathrm{LERP}(\lambda, p, q) \mid \lambda \in \mathbb{R}\}$, with $\mathrm{LERP}(\lambda, p, q) = \nabla F^{-1}((1-\lambda)\nabla F(p) + \lambda \nabla F(q))$; see [10, 5, 11]. To find the Bregman projection $q_B = \arg\min_{x \in B(c,R)} D_F(x||c)$ of a query point q onto a Bregman ball $B(c, R)$, we first check whether q is outside the ball or not: $D_F(q||c) > R$. If not, $D_F(q||c) \leq R$ and the projection is simply the point itself: $q_B = q$. Now consider the geodesic segment Γ_{cq}, the projection point q_B belongs necessarily to the geodesic Γ_{cq}: $q_B = \mathrm{LERP}(\lambda_q, c, q)$ for some $\lambda_q \in [0, 1]$. The value λ_q is approximated by a bisection search as follows. Initially, let $\lambda_m^{(0)} = 0$ and $\lambda_M^{(0)} = 1$ and consider a midpoint $m = \mathrm{LERP}(\lambda^{(0)}, c, q)$ obtained for $\lambda^{(0)} = \frac{\lambda_m^{(0)} + \lambda_M^{(0)}}{2}$. If $D_F(m||c) > R$ then recurse on $[\lambda_m^{(1)}, \lambda_M^{(1)}] = [\lambda_m^{(0)}, \lambda^{(0)}]$, otherwise recurse on $[\lambda_m^{(1)}, \lambda_M^{(1)}] = [\lambda^{(0)}, \lambda_M^{(0)}]$ and so on until the range size $\lambda_M^{(i)} - \lambda_m^{(i)}$ goes below a threshold ϵ (typically $\epsilon \in [10^{-10}, 10^{-5}]$). This yields a fine approximation of $q_B \sim \mathrm{LERP}(\lambda^{(i)}, c, q)$ by splitting the "λ" intervals a dozen of times. (Recall that the algorithm keeps a Bregman annulus, and refine the inner/outer radii to decide whether to prune or explore a sub-tree.)

2.5 Handling symmetrized Bregman divergences

Many content-based information retrieval systems (CBIRs) need *symmetrical* distortions measures. Except for generalized quadratic distances (Mahalanobis), the symmetrized Bregman divergence is technically not a Bregman divergence [10]. The symmetric KL divergence (SKL) goes by the name of Jensen-Shannon $\mathrm{JS}(p;q) = \frac{1}{2}\mathrm{KL}(p||\frac{p+q}{2}) + \frac{1}{2}\mathrm{KL}(q||\frac{p+q}{2})$. It turns out that these symmetrized Bregman divergences [11] are all generalizations of the Jensen remainder obtained for convex generators F: $\mathrm{JS}_F(p;q) = \frac{1}{2}(F(p) + F(q)) - F(\frac{p+q}{2})$. (Besides the Jensen-Shannon divergence for $F(x) = x \log x$, these so-called Burbea-Rao divergences also includes the important COSH distance used in speech/sound processing for the Burg entropy $F(x) = -\log x$.)

Bb-tree construction ($bs = 50$)					
method	*iter*	*depth*	$depth_{avg}$	*Leav.*	*speed-up*
2-m.	10	53	28.57	594	1
2-m.++	10	58.33	31.18	647	1.03
2-m.++	**0**	**20**	**10.76**	**362**	**19.71**

Table 1: Construction time of Bregman ball trees.

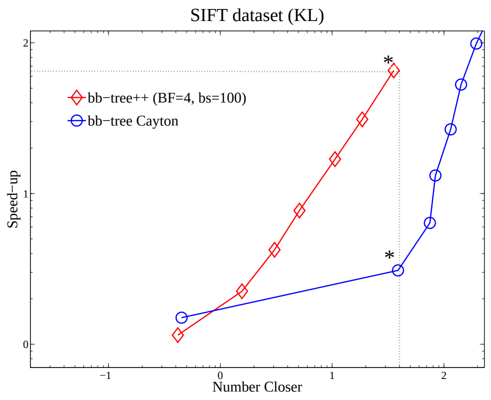


Figure 1: BB-tree versus BB-tree++ (log-log plot scale). The axis denotes the error rate expressed as the number of closer points to the approximated NN.

The symmetrized Bregman centroid can be explicitly computed following the geodesic-walk algorithm of [11].

3 Experiments

Table 1 compares different partitioning methods when building a Bb-tree on the SIFT dataset [13] (10,000 visual feature points extracted from images, encoded in dimension 1111). Bregman 2-`means++` with $iter = 0$ means that only the initialization step is carried out. The speed-up is by comparison with the most computationally expensive k-means method.

We used the maximum number of explored leaves as a parameter for stopping the branch-and-bound search. In each experiment of approximate search, we fixed a value of this parameter (from *near-exact* search to visiting only a *single* leaf), then we evaluated error rate and computational cost. Namely, the error rate is expressed as the number of closer points to the approximated NN (a quantity called Number Closer), while the speed-up is given by the ratio between the number of divergence computations in Bb-tree over the brute-force search. Figure 1 displays a comparison of Cayton's Bb-tree with our Bb-tree++ in approximate search on SIFT dataset (log-log plot).

Figure 2 presents the results of approximate search queries on SIFT dataset wrt. SKL divergence (symmetrized Bregman). The marked point shows a speed-up of 60.3 ($10^{1.78}$) when the Number Closer equals 14 ($10^{1.15}$), i.e. 0.14% of database points (log-log plot).

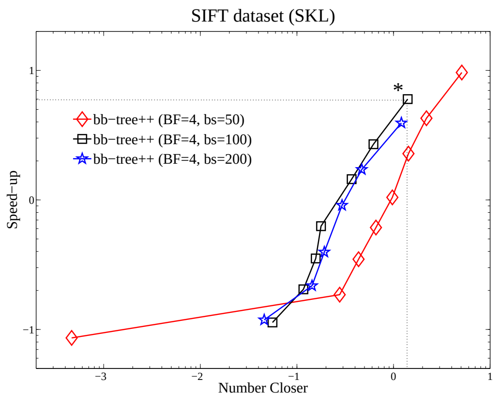


Figure 2: NN queries with respect to Jensen-Shannon divergence (symmetric Kullback-Leibler, SKL).

Acknowledgments
We thank L. Cayton for sharing with us his code and SIFT high-dimensional datasets.

References

[1] S.-I. Amari and N. Nagaoka. *Methods of Information Geometry*. Oxford University Press, 2000.

[2] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.

[3] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, pages 327–336, 2008.

[4] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research (JMLR)*, 6:1705–1749, 2005.

[5] L. Cayton. Fast nearest neighbor retrieval for Bregman divergences. In *ICML*, pp. 112–119, 2008.

[6] B. Chazelle. Technical perspective: finding a good neighbor, near and fast. *Commun. ACM*, 51(1):115, 2008.

[7] G. Hamerly and C. Elkan. Learning the k in k-means. In *NIPS*, 2003.

[8] N. Kumar, L. Zhang, and S. K. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *ECCV*, pp. 364–378, 2008.

[9] B. Manthey and H. Röglin. Improved Smoothed Analysis of the k-Means Method. In *SODA* (arXiv:0809.1715), 2009.

[10] F. Nielsen, J.-D. Boissonnat, and R. Nock. On Bregman Voronoi diagrams. In *SODA*, pp. 746–755, 2007.

[11] F. Nielsen and R. Nock. Bregman sided and symmetrized centroids. In M. Ejiri, R. Kasturi, editor, *International Conference on Pattern Recognition (ICPR)*. IEEE CS Press, 2008. (arXiv:0711.3242)

[12] R. Nock, P. Luosto, and J. Kivinen. Mixed Bregman clustering with approximation guarantees. In *ECML/PKDD (2)*, pages 154–169, 2008.

[13] D. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, pages 91–111, Vol. 60, 2005.

Maintaining Exactly the Convex Hull of Points Moving along Circles

Paul-Georg Becker [*] Elvir Büyükbayrak [†] Nicola Wolpert [‡]

Abstract

In this work we consider the problem of maintaining geometrically exact the convex hull of points moving along circles in the plane. The points are allowed to start with an arbitrary rational angle (in the radian system) on the circle. The main difficulty with respect to the aim of exact geometric computing in this setting is that we have to deal with transcendental numbers. The points in time where the convex hull change are not algebraic but transcendental. We show that, even in this non-algebraic setting, all certificates in the underlying kinetic convex hull algorithm [1] can be evaluated mathematically exact. Our approach has been implemented prototypically.

1 Introduction

In this work we consider n points moving along circles in the Euclidean plane. The points are allowed to start with an arbitrary rational angle (in the radian system) on the circle.The motion of point P_i, depending on the time t, is described by its parameterization

$$P_i(t) = \begin{pmatrix} x_i(t) \\ y_i(t) \end{pmatrix} = \begin{pmatrix} r_i \cdot \cos(t - t_i) + a_i \\ r_i \cdot \sin(t - t_i) + b_i \end{pmatrix}.$$

We assume that all r_i, a_i, b_i and t_i are rational numbers. Due to the parameterization, point P_i moves along the circle with center (a_i, b_i) and radius r_i. At time $t = 0$ it starts its movement at the point

$$\begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} r_i \cdot \cos(t_i) + a_i \\ -r_i \cdot \sin(t_i) + b_i \end{pmatrix}.$$

In this work we only consider the case that all points move with the same speed in the sense that they all need the same time to circle once.

For the n moving points we want to compute and maintain their convex hull. The aim is to do this in the sense of the paradigm of exact geometric computing (EGC). Every change in the convex hull should be determined mathematically exact without any numerical errors. The main difficulty with respect to the goal of exact geometric computing in our setting is that we have to deal with transcendental numbers. The points in time t where the convex hull change are not algebraic but transcendental. We will derive that every t is the root of a degree-two polynomial $at^2 + bt + c$. In this polynomial all three coefficients a, b and c are of the form $\sum_{j=1}^{n} q_j \sin(\alpha_j) + p_j \cos(\alpha_j)$ with $p_j, q_j, \alpha_j \in \mathbb{Q}$. We show that, even in this non-algebraic setting where the coefficients a, b, c are transcendental numbers, all certificates in the kinetic convex hull algorithm of Razazzi and Sajedi [1] can be evaluated mathematically exact.

In the last years a lot of effort was undertaken to advance EGC. Most of the problems considered so far in this context are static ones, like for example computing arrangements of algebraic curves and surfaces [5] or Voronoi-diagrams of line segments [7], just to mention a few. All these problems have in common that one main task is to compute exactly with algebraic numbers, i.e. roots of rational polynomials. Recently, Russel et al [6] presented a work on extending the results made in EGC for static geometric algorithms to algorithms involving motion. Also here, on the numerical side, the main task is to perform the arithmetic with algebraic numbers exactly and efficiently. The only work we know which deals with a geometric problem involving transcendental numbers is the one by Chang et al [4]. They show that the shortest path amidst disc obstacles is computable.

2 The Underlying Kinetic Algorithm

The underlying kinetic algorithm we use for maintaining the convex hull of a set S of n moving points is the one proposed by Razazzi and Sajedi [1]. In an initialization step at time $t = 0$ a sequence $C_1, C_2, \ldots$ of nested convex hulls is computed. The outermost convex hull $C_1 = ch(S_1)$ is just the convex hull of the points in $S_1 := S$. The i-th convex hull $C_i = ch(S_i)$ is the one of the remaining points $S_i := S_{i-1} \backslash ch(S_{i-1})$, if S_i is not empty. For every point p on every convex hull C_i two references are stored: one to the *previous* and one to the *next* point on the same convex hull C_i. Two consecutive hulls C_{i+1} and C_i are also connected: Every point p of C_{i+1} stores one reference to the *first* point of C_i which is visible from p and one reference to the *last* visible point. Consider point B_3 in Figure 1. B_3 lies on the convex hull B and it is $B_3.previous = B_2$ and $B_3.next = B_4$. B_3 can see the part of the convex hull C between the points C_2 and

[*]University of Applied Sciences Stuttgart, paul-georg.becker@hft-stuttgart.de
[†]University of Applied Sciences Stuttgart, elvir.bb@gmx.net
[‡]University of Applied Sciences Stuttgart, nicola.wolpert@hft-stuttgart.de

C_4 and therefore $B_3.first = C_2$ and $B_3.last = C_4$.

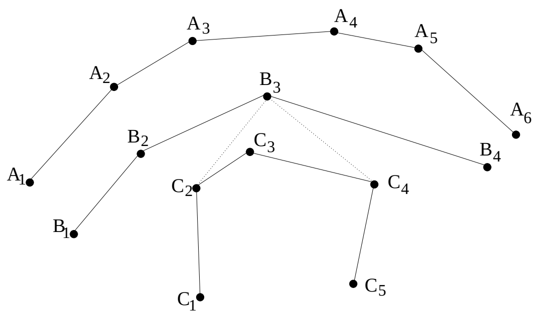


Figure 1: Three nested convex hulls A, B, and C

Now the points start moving and at some points in time the hulls C_i will change. These are called the *events*. Changes in the outermost convex hull C_1 of course change our result. In order to notice every event, a set of certificates is computed for every point. The arrival of such a certificate indicates an event and will cause some recomputation. We explain all necessary certificates with respect to the point B_3 in Figure 1:

- **GoIn(B_3):** B_3 moves towards C and leaves B $\Leftrightarrow$ B_3, B_2 and B_4 are collinear.
- **GoOut(B_3):** B_3 moves towards A and leaves B $\Leftrightarrow$ there exists a point A_x with $A_x.first = B_3$ and B_3, A_x and A_{x-1} are collinear.
- **NotFirstChild(B_3):** C_2 is no longer visible from B_3 $\Leftrightarrow$ C_3, B_3 and C_2 are collinear.
- **BeFirstChild(B_3):** C_1 becomes visible from B_3 $\Leftrightarrow$ C_1, B_3 and C_2 are collinear.
- **NotLastChild(B_3):** C_4 is no longer visible from B_3 $\Leftrightarrow$ C_3, B_3 and C_4 are collinear.
- **BeLastChild(B_3):** C_5 becomes visible from B_3 $\Leftrightarrow$ C_5, B_3 and C_4 are collinear.

The arrival times for every certificate are stored in a priority queue. Our aim is to design an exact kinetic algorithm. This means that we want to compute correctly the arrival times of the events and we do not want to miss any event due to numerical errors. We also want to sort the arrival times correctly in the priority queue so that the events can be executed in the right chronological order. From the certificates above one can derive that the two main task to be solved are to

a) determine exactly the point in time, when three points become collinear (called *collinearity-test*) and to

b) compare exactly two such points in time in order to sort the events in the right order in the priority queue (called *comparison-operation*).

Regarding the recomputation of the nested convex hulls we will not give the details here but refer to the paper of Razazzi and Sajedi [1]. Numerically, the main task is to recompute the first- and last-pointers. This can be realized at rational points in time again with the help of the collinearity-test and the comparison-operation.

3 Collinearity-Test

We have seen that one main computation in the underlying kinetic algorithm is the test whether three points become collinear. It is well known that collinearity of three points $P_i(t)$, $P_j(t)$, and $P_k(t)$ in the plane can be determined by evaluating the determinant

$$\det(P_i(t), P_j(t), P_k(t)) = \det \begin{pmatrix} 1 & x_i(t) & y_i(t) \\ 1 & x_j(t) & y_j(t) \\ 1 & x_k(t) & y_k(t) \end{pmatrix}.$$

Computing the determinant leads to an expression in the variable t. In order to find the time where the three points become collinear, we have to look for the roots of this expression. Unfortunately, due to our parameterization, we have terms of the form $\sin(t-t_i)$ and $\cos(t-t_i)$ in the expression. To eliminate them we first apply the addition-theorems

$$\begin{aligned} \sin(t-t_i) &= \sin(t)\cos(t_i) - \sin(t)\cos(t_i) \\ \cos(t-t_i) &= \cos(t)\cos(t_i) + \sin(t)\sin(t_i) \end{aligned}$$

and then do the substitution

$$\cos(t) := \frac{1-t'^2}{1+t'^2} \quad \text{and} \quad \sin(t) := \frac{2t'}{1+t'^2}.$$

To ease readability we will again write t instead of t' in the following. We define the set

$$T := \{\sum_{j=1}^{n} q_j \sin(\alpha_j) + p_j \cos(\alpha_j) \mid n \in \mathbb{N}, \\ q_j, p_j, \alpha_j \in \mathbb{Q}, \alpha_j > 0, \alpha_i \neq \alpha_j \text{ for } i \neq j\}.$$

Applying the addition-theorems, substituting the cosine- and sine-terms and then doing some calculations (which we omit in this presentation) lead to the result that the determinant always has the form

$$\frac{at^2 + bt + c}{1 + t^2}$$

with $a, b, c \in T$. We are interested in the roots of this expressions, that means we are interested in the roots of the polynomial $p(t) = at^2 + bt + c$ of degree 2 in t

with coefficients $a, b, c \in T$. If $a \neq 0$, the two roots of p are

$$\frac{-b+\sqrt{b^2-4ac}}{2a} \quad \text{and} \quad \frac{-b-\sqrt{b^2-4ac}}{2a}.$$

The discriminant $b^2 - 4ac$ is again an element of T:

Lemma 1 *If $a, b \in T$ then $a \cdot b \in T$.*

Proof. The lemma follows directly from the rules for the sine- and cosine-functions:

$$\begin{aligned}
\sin x \sin y &= 1/2 \cdot (\cos(x-y) - \cos(x+y)) \\
\cos x \cos y &= 1/2 \cdot (\cos(x-y) + \cos(x+y)) \\
\sin x \cos y &= 1/2 \cdot (\sin(x-y) + \sin(x+y))
\end{aligned}$$

□

How can we decide whether a is equal to zero or not? To answer this question we need a definition and the Theorem of Hermite-Lindemann, which are both well known from number theory:

Definition 1 *Let $\alpha_1, \ldots, \alpha_n$ be real numbers. They are called* $\mathbb{Q}$-linearly independent *iff there exist no rational numbers $q_1, \ldots, q_n$, not all equal to zero, with $\alpha_1 q_1 + \cdots + \alpha_n q_n = 0$.*

Theorem 2 *(Hermite-Lindemann) If α is a non-zero algebraic number (i.e. a number, which is the root of a rational polynomial), then e^α is transcendental (i.e. not algebraic).*

We can proof the following

Corollary 3 *If $\alpha_1, \ldots, \alpha_n$ are pairwise distinct positive rational numbers, then the $2n$ real numbers $\sin(\alpha_1), \ldots, \sin(\alpha_n), \cos(\alpha_1), \ldots, \cos(\alpha_n)$ are $\mathbb{Q}$-linearly independent.*

Proof. Assume, there exist $q_1, \ldots, q_n, p_1, \ldots, p_n \in \mathbb{Q}$ with $\sum_{j=1}^n q_j \sin(\alpha_j) + p_j \cos(\alpha_j) = 0$. Substituting $\sin(\alpha_j) = \frac{1}{2i}(e^{i\alpha_j} - e^{-i\alpha_j})$, $\cos(\alpha_j) = \frac{1}{2}(e^{i\alpha_j} + e^{-i\alpha_j})$ leads to

$$\sum_{j=1}^n (p_j - iq_j)e^{i\alpha_j} + \sum_{j=1}^n (p_j + iq_j)e^{-i\alpha_j} = 0.$$

Let $A \in \mathbb{N}$ be the common denominator of $\alpha_1, \ldots, \alpha_n$ (i.e. $A\alpha_j \in \mathbb{Z}$) and let $M := \max\{A\alpha_1, \ldots, A\alpha_n\}$. Now consider the polynomial

$$P(X) = \sum_{j=1}^n (p_j - iq_j)X^{M+A\alpha_j} + \sum_{j=1}^n (p_j + iq_j)X^{M-A\alpha_j}.$$

The coefficients of P are numbers from the field $\mathbb{Q}(i)$. We know that $P(e^{i/A}) = 0$. Due to the Theorem of Hermite-Lindemann, $e^{i/A}$ is transcendental over $\mathbb{Q}$ and therefore also transcendental over $\mathbb{Q}(i)$. It follows that P has to be the zero-polynomial. Since all the α_j are pairwise distinct, all the exponents $M + A\alpha_j$ and $M - A\alpha_j$ are pairwise distinct, and therefore all coefficients $p_j - iq_j$ and $p_j + iq_j$ of P have to be zero. It follows directly that all the p_j and q_j have to be zero, proving the corollary. □

From Corollary 3 one can derive directly a criterion for testing whether an element $a \in T$ is equal to zero:

Corollary 4 *An element $a = \sum_{j=1}^n q_j \sin(\alpha_j) + p_j \cos(\alpha_j) \in T$ is equal to zero iff $q_j = p_j = 0$ for all $1 \leq j \leq n$.*

4 Comparison-Operation

The second main task is to compare two of the degree-two roots in order to sort them in the correct order into the priority queue. More formally: Given

$$x_1 = \frac{-b_1 \pm \sqrt{b_1^2 - 4a_1c_1}}{2a_1}, \quad x_2 = \frac{-b_2 \pm \sqrt{b_2^2 - 4a_2c_2}}{2a_2}$$

with $a_1, a_2, b_1, b_2, c_1, c_2 \in T$, decide whether $x_1 < x_2$. We assume for the moment that we have a black-box *sign* which computes the sign of an arbitrary number $a \in T$. We will explain in the next section how to realize this black-box. For simplicity we set $p_i = b_i^2 - 4a_ic_i$ for $i = 1, 2$ and assume that $a_1, a_2 > 0$ (otherwise multiply by -1). To exclude complex roots we test $sign(p_i) > 0$. If we have two real roots x_1 and x_2 this leads to

$$\begin{aligned}
x_1 < x_2 \quad &\Leftrightarrow \quad \frac{-b_1 \pm \sqrt{p1}}{2a_1} < \frac{-b_2 \pm \sqrt{p2}}{2a_2} \\
&\Leftrightarrow \quad a_2(-b_1 \pm \sqrt{p_1}) < a_1(-b_2 \pm \sqrt{p_2}) \\
&\Leftrightarrow \quad a_1b_2 - a_2b_1 < \pm a_1\sqrt{p_2} \mp a_2\sqrt{p_1}.
\end{aligned}$$

Since, due to Lemma 1, $a_1b_2 - a_2b_1 \in T$, we can use the black-box and compute $sign(a_1b_2 - a_2b_1)$. Next we determine the sign of the right side. For readability we substitute $q := \pm a_1$ and $p := \mp a_2$. We are interested in the sign of $p\sqrt{p_1} + q\sqrt{p_2}$ with $p, q, p_1, p_2 \in T$. If $sign(p)$ and $sign(q)$ are equal, we directly know the sign of $p\sqrt{p_1} + q\sqrt{p_2}$. If the two signs are different, we have to work a bit more: Assume that p is positive and q is negative (the other case works analogously). Then

$$\begin{aligned}
&p\sqrt{p_1} + q\sqrt{p_2} \text{ has a positive sign} \\
\Leftrightarrow \quad &|p|\sqrt{p_1} > |q|\sqrt{p_2} \\
\Leftrightarrow \quad &p^2 \cdot p_1 > q^2 \cdot p_2 \\
\Leftrightarrow \quad &sign(p^2 \cdot p_1 - q^2 \cdot p_2) > 0.
\end{aligned}$$

Now we know the signs of both sides of the inequality $a_1b_2 - a_2b_1 < p\sqrt{p_1} + q\sqrt{p_2}$. If they are different, we can decide whether the inequality is true or not. Otherwise we compare $|a_1b_2 - a_2b_1|$ and $|p\sqrt{p_1} + q\sqrt{p_2}|$:

$$\begin{aligned}
&|a_1b_2 - a_2b_1| < |p\sqrt{p_1} + q\sqrt{p_2}| \\
&\quad\Leftrightarrow\quad (a_1b_2 - a_2b_1)^2 < (p\sqrt{p_1} + q\sqrt{p_2})^2 \\
&\quad\Leftrightarrow\quad (a_1b_2 - a_2b_1)^2 < p^2p_1 + q^2p_2 + 2pq\sqrt{p_1}\sqrt{p_2} \\
&\quad\Leftrightarrow\quad (a_1b_2 - a_2b_1)^2 - p^2p_1 - q^2p_2 < 2pq\sqrt{p_1}\sqrt{p_2}.
\end{aligned}$$

Again, if $sign((a_1b_2 - a_2b_1)^2 - p^2p_1 - q^2p_2)$ and $sign(pq)$ are different, we are done. Otherwise we compare $|(a_1b_2 - a_2b_1)^2 - p^2p_1 - q^2p_2|$ and $|2pq\sqrt{p_1}\sqrt{p_2}|$:

$$\begin{aligned}
&|(a_1b_2 - a_2b_1)^2 - p^2p_1 - q^2p_2| < |2pq\sqrt{p_1}\sqrt{p_2}| \\
&\quad\Leftrightarrow\quad ((a_1b_2 - a_2b_1)^2 - p^2p_1 - q^2p_2)^2 \\
&\qquad\qquad < 4p^2q^2p_1 \cdot p_2 \\
&\quad\Leftrightarrow\quad ((a_1b_2 - a_2b_1)^2 - p^2p_1 - q^2p_2)^2 \\
&\qquad\qquad - 4p^2q^2p_1 \cdot p_2 < 0.
\end{aligned}$$

Now, due to Lemma 1, the left side of the inequality is an element of T and we can finally compute its sign.

5 Black-Box

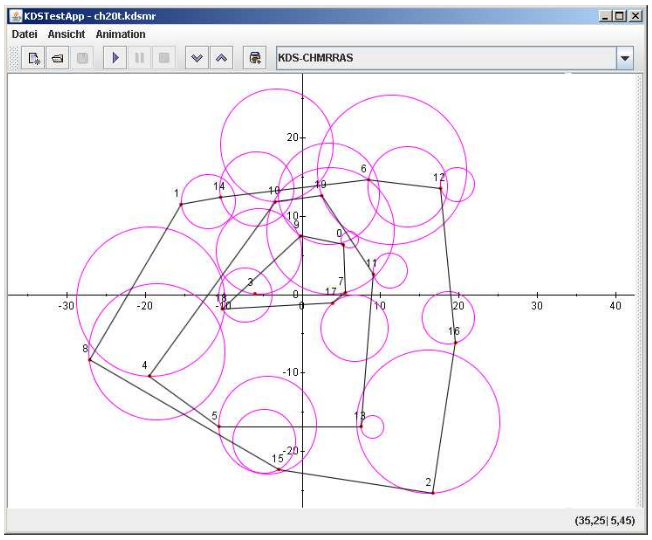
What remains to do is to implement the black-box. Given an element $a = \sum_{j=1}^{n} q_j \sin(\alpha_j) + p_j \cos(\alpha_j) \in T$ we want to determine $sign(a)$. From Corollary 4 we know that $a = 0$ iff $p_j = q_j = 0$ for all $1 \leq j \leq n$. Assume $a \neq 0$. We propose a method that starts with a given precision ϵ and adaptively increases the precision until the sign of a can be determined precisely. For the given precision ϵ we compute for all $1 \leq j \leq n$, e.g. using the Apfloat-library [2] or the Core-Library [3], approximations s_j for $\sin(\alpha_j)$ and c_j for $\cos(\alpha_j)$ such that $|s_j - \sin(\alpha_j)| < \epsilon$ and $|c_j - \cos(\alpha_j)| < \epsilon$. Based on these approximations we compute the approximation $\tilde{a} = \sum_{j=1}^{n} q_j s_j + p_j c_j$ of a. It is

$$\begin{aligned}
|a - \tilde{a}| &= |\sum_{j=1}^{n} q_j(\sin(\alpha_j) - s_j) + p_j(\cos(\alpha_j) - c_j)| \\
&\leq \sum_{j=1}^{n} |q_j| \cdot |\sin(\alpha_j) - s_j| + |p_j| \cdot |\cos(\alpha_j) - c_j| \\
&< \epsilon \cdot \sum_{j=1}^{n} |q_j| + |p_j| \;=:\; \delta_\epsilon.
\end{aligned}$$

If $|\tilde{a}| > \delta_\epsilon$, we can read off the sign of a by looking at the sign of $\tilde{a}$. Otherwise we start the computation again with a smaller ϵ. The procedure terminates because $a \neq 0$ and $\tilde{a}$ converges towards a.

6 Implementation

We have a prototypical implementation of the kinetic algorithm including the exact mathematical evaluation of the certificates described before. The implementation is done in Java. The black-box is implemented using the Apfloat-Library [2]. As an input one can interactively define circles. During the execution, the program renews the output, i.e. the nested convex hulls, regularly every 50 milliseconds. All the events which occurred during two such time-slots are computed exactly in the right order and reported. See below a screenshot taken during the execution of the program.



References

[1] M. R. Razzazi, A. Sajedi. *Kinetic Convex Hull Maintenance Using Nested Convex Hulls.* EuroCG 2003.

[2] M. Tommila. *Apfloat.* `http://www.apfloat.org/`.

[3] C. Yap. *CORE.* `http://cs.nyu.edu/exact/core_pages/index.html`.

[4] E. Chang, S.W. Choi, D. Kwon, H. Park, C. Yap. *Shortest Path amidst Disc Obstacles is Computable* Special Issue on Geometric Constraints, Intl.J.Comp.Geom. & Applic., 16:5-6(2006)567–590 .

[5] J.-D. Boissonat, M. Teillaud (editors). *Effective Computational Geometry for Curves and Surfaces* Springer, 2006.

[6] D. Russel, M. I. Karavelas and L. J. Guibas. *A package for Exact Kinetic Data Structures and Sweepline Algorithms.* Computational Geometry: Theory and Applications, Special Issue on CGAL, 38(1-2):111-127, September 2007.

[7] H. Everett, D. Lazard, S. Lazard, M. S. El Din. *The Voronoi Diagram of Three Lines.* 23th ACM Symposium on Computational Geometry, 2007.

k-Means Motion Clustering

Frank Hellweg[*] Christian Sohler[†]

Abstract

We present algorithms for k-Means Motion Clustering, where the problem of motion clustering is formalized as follows: The objects to cluster are n points in the d-dimensional real space, and their motion is given by functions that are polynomials of the time whose degree is bounded by some constant μ. The objective is to compute a clustering that guarantees a (bicriteria) $\mathcal{O}(1)$-approximation to the optimal k-clustering at any time, without recalculating it at certain points of time. Our result extends previous work by Har-Peled [9] for the k-center objective function to the k-means objective function.

1 Introduction

Clustering is the process of partitioning a given point set into subsets such that the points within a subset are close to each other and objects in different subsets are distant. In this paper we consider clustering of moving points. A clustering of moving points can, for example, be used to maintain a hierarchical structure in mobile networks. Such a hierarchical structure can then be used for routing, data management, etc. We will assume that the motion of the points is known in advance and we attempt to compute a clustering that is good for the whole motion.
We will measure the quality of the clustering in terms of the k-means objective function, i.e. we compute k moving points (the cluster centers) and , at any fixed point of time, the quality of the clustering induced by these points is the sum of the squared distances to the nearest center. In the context of mobile networks this quality measure can be viewed as the average energy needed to connect to the cluster center. Notice, however, that the cluster center is not necessarily part of the moving point set.
Our result extends previous work by Har-Peled [9] for the k-center objective function to the k-means objective function. Technical details that are left out here can be found in [6].

Related Work For motion clustering with the objective to only compute a clustering once for all time points, there are mainly two results, the first being a result of Bespamyatnikh et al. [3], who, among other things, analyze the 1-median clustering problem in the euclidean plane for arbitrary motion functions, considering Euclidean distances. By reducing to the Manhattan distances version, they get a $\sqrt{2}(2-2/n)$-approximation algorithm for this problem.
The second main result is the mentioned result of Har-Peled [9]: He consideres k-center motion clustering of points whose motion is described by polynomials in the time, with a degree bounded by some constant μ, and one of his results is an algorithm that computes a motion clustering in running time $\mathcal{O}(nk)$ using $(15k)^{\mu+1}$ cluster centers and guaranteeing an approximation factor of $3^{\mu+1}-1$ at any point of time. He also proves that one needs at least $k^{\mu+1}$ cluster centers to guarantee an $\mathcal{O}(1)$-approximation for any point of time. His worst-case example family also holds for k-means clustering.
His basic idea is to solve a subproblem he calls *stabbing*, which is essentially a clustering problem which relaxes the restriction that all clusters have to be at the same point of time. By solving this problem, he gets k clusters that are good for some point of time each, and he uses this for decreasing the degree of the motion functions by one and recursively calculating motion clusterings for the k subproblems defined by the k clusters and the new motion functions. For $\mu = 0$, it suffices to compute an approximate standard k-center clustering, and Har-Peled shows that one can maintain an approximation factor throughout all the recursions back to the original problem.
There are also some results for motion clustering and related problems, e.g. facility location, that use Kinetic Data Structures to maintain an approximate clustering instead of calculating it once and using it for all points of time; see [4, 5, 7, 8, 10].

Preliminaries In this section we will define the k-means motion clustering problem formally. A (polynomially) moving point is a mapping $p : \mathbb{R} \to \mathbb{R}^d$, whereas p is polynomial in the time parameter t with a maximum degree of μ. Let $p(t)_i$ denote the i-th coordinate of p at the point of time t. For a set $P = \{p_1, \ldots, p_n\}$ of polynomially moving points let $P[t] := \bigcup_{p \in P}\{p(t)\}$ denote the set of static points at time t.
A k-clustering of such a set P is a partition $C = \{C_1, \ldots C_k\}$ of P such that $\biguplus_{1 \le i \le k} C_i = P$. The cost

[*]Department of Computer Science, University of Bonn, hellweg@informatik.uni-bonn.de
[†]Department of Computer Science, University of Bonn, sohler@informatik.uni-bonn.de

of such a clustering C for P at the point of time t are defined as $\gamma(C,P,t) = \sum_{1\le i\le k}\sum_{p_j\in C_i} ||p_j[t] - \hat{c}_i(t)||^2$, whereas $\hat{c}_i(t)$ denotes the centroid of the cluster C_i at the point of time t; it is well-known that the centroid of a point set is the optimum solution for the 1-means problem for that point set.
The cost of an optimal k-clustering of P at the point of time t are denoted as $\gamma_{opt}(P,k,t) = min_{\hat{C}=\{\hat{C}_1,\dots,\hat{C}_k\}:\biguplus_i \hat{C}_i=P}\gamma(\hat{C},P,t)$ and the cost of an optimum k-clustering for any point of time as $\gamma_{opt}(P,k) = \min_t \gamma_{opt}(P,k,t)$. For static point sets we define the cost functions in the same way, ommitting the parameter t. We will, slightly abusing the notation, speak of a clustering C for $P[t]$ if we mean the clustering C for P at the point of time t.
We call a clustering C of a set P of polynomially moving points a (c,m,k)-static clustering if it consists of at most m clusters and for each point of time t yields a c-approximation to the optimum k-clustering of that point of time, i.e. $\gamma(C,P,t) \le c\cdot\gamma_{opt}(P,k,t)\forall t$.
We need to solve an auxilliary problem that we call k-means stabbing, following Har-Peled's notation. A k-stabbing (S,t^*) of a moving point set P consists of a partition $S = \{S_1,\dots S_k\}$ of P into k disjoint sets, which we call *stabbing clusters*, and a function $t^* : S \to \mathbb{R}$ that assigns a point of time to each stabbing cluster. The cost for a stabbing (S,t^*) are defined as $\gamma_{stab}(S,t^*,P) := \sum_{T\in S}\gamma(\{T\},T,t^*(T))$. A stabbing (S,t^*) of a set P of moving points is called (c,m,k)-Stabbing if it consists of at most m stabbing clusters and satisfies $\gamma_{stab}(S,t^*) \le c\cdot\gamma_{opt}(P,k)$, i.e. has at most the cost of the optimal clustering of any point of time.

2 An Algorithm for k-Means-Stabbing

In this section we describe how to compute a k-means motion clustering by solving several stabbing problems; after that we will present an algorithm for these stabbing problems. The basic idea is the same as Har-Peled's, which we described in Section 1; for a moving point set P with a degree of motion of at most μ and $k\in\mathbb{N}$, the algorithm works as follows: Compute a (c,l,k)-stabbing (S,t^*) of P, whereas $l\ge k$ and c is a constant, and look at each stabbing cluster seperately. For each stabbing cluster $T\in S$, move the curves of the moving points in T, such that they intersect in the centroid of $T[t^*(T)]$, getting the point set T'. Now consider the centroid of $T[t^*(T)]$ being the center of the coordinate system: The motion functions of all the points in T' are now polynomials without a constant summand, so we can divide them by their parameter t, getting a set of points with a degree of motion of at most $\mu-1$. Now we recursively cluster this set of moving points. After we have done this for all stabbing clusters $T\in S$, each recursive call retrieving a motion clustering $C_{T'}$, we return the motion clustering $C = \bigcup_{T\in S} C_T$, whereas C_T is the clustering corresponding to $C_{T'}$ for all $T\in S$.
It remains to show that an approximation factor can be maintained while reversing these steps. First we will show that an approximation factor can be maintained when moving the points' curves back to their original positions.

Lemma 1 *Let $P = \{p_1,\dots,p_n\}$ and $P' = \{p'_1,\dots,p'_n\}$ be sets of n static points, whereas for $i\in\{1,\dots,n\}$ p'_i is created from p_i by moving p_i by a Euclidean distance of v_i. Let $\sum_{1\le i\le n} v_i^2 \le \rho\cdot\gamma_{opt}(P,k)$ for a constant $\rho\in\mathbb{R}^+$. Then, for all $k\in\mathbb{N}$ holds $\gamma_{opt}(P',k) \le (1+\sqrt{\rho})^2\gamma_{opt}(P,k)$.*

Proof. Let U be an optimal k-clustering for P and d_i the Euclidean distance from p_i to his cluster center in U for all i. Let U' be the clustering of P' corresponding to U; note that U' is not necessarily optimal. We can bound the cost of an optimal k-clustering of P' by $\gamma_{opt}(P',k) \le \gamma(U',P') \le \sum_{C\in U'}\sum_{p_i\in C}(d_i+v_i)^2$, the last inequality following from the fact that choosing the optimal center for each cluster in U' yields at most the cost as choosing the optimal center of the according cluster in U; the later is at most $\sum_{p_i\in C}(d_i+v_i)^2$ for each cluster $C\in U'$ by the triangle inequality. We conclude

$$\begin{aligned}\gamma_{opt}(P',k) &\le \sum_{C\in U'}\sum_{p_i\in C} d_i^2 + 2d_iv_i + v_i^2\\ &\le \sum_{p_i\in P} d_i^2 + \sum_{p_i\in P} v_i^2 + 2\sqrt{\sum_{p_i\in P} d_i^2}\sqrt{\sum_{p_i\in P} v_i^2}\\ &\le \gamma_{opt}(P,k) + \rho\cdot\gamma_{opt}(P,k) + 2\sqrt{\rho}\cdot\gamma_{opt}(P,k)\\ &= (1+\sqrt{\rho})^2\gamma_{opt}(P,k),\end{aligned}$$

whereas the second inequality follows from the Cauchy-Schwarz Inequality. □

Lemma 2 *Let $P = \{p_1,\dots,p_n\}$ and $P' = \{p'_1,\dots,p'_n\}$ be sets of n moving points, whereas for $i\in\{1,\dots,n\}$ p'_i is created from p_i by moving the curve of p_i by an euclidean distance of v_i. Let $\sum_{1\le i\le n} v_i^2 \le \rho\cdot\gamma_{opt}(P,k)$ for a constant $\rho\in\mathbb{R}^+$. For any (c,m,k)-static clustering T' of P' the corresponding clustering T of P is a $((\sqrt{c}(1+\sqrt{\rho})+\sqrt{\rho})^2,m,k)$-static clustering.*

Proof. Let t be a fixed but arbitrary point of time. For $p_i\in P'$ let d_i be the Euclidean Distance from $p'_i[t]$ to its cluster center in T' at time t. We can bound the cost of T by $\gamma(T,P,t) \le \sum_{C\in T}\sum_{p_i\in C}(d_i+v_i)^2$ using a similar argument as in the proof of Lemma 1. We conclude

$$\begin{aligned}\gamma(T,P,t) &\le \sum_{p_i\in P} d_i^2 + \sum_{p_i\in P} v_i^2 + \sqrt{\sum_{p_i\in P} d_i^2}\sqrt{\sum_{p_i\in P} v_i^2}\\ &\le (\sqrt{c}(1+\sqrt{\rho})+\sqrt{\rho})^2\cdot\gamma_{opt}(P,k,t)\end{aligned}$$

due to Lemma 1, and so T is a $((\sqrt{c}(1+\sqrt{\rho})+\sqrt{\rho})^2, m, k)$-static clustering of P. □

The next Lemma shows that if we compute an approximate motion clustering on a set of moving points and then multiply all motion functions by their parameter t, the corresponding clustering on the resulting point set will guarantee the same approximation factor:

Lemma 3 *Let $P = \{p_1, \dots, p_n\}$ be a set of n polynomially moving points with a degree of motion of at most μ and $p_i[0] = \vec{0}$ for $1 \leq i \leq n$. Then, for the mapping $f(p(t)) := p(t)/t$ and $P' = \{p'_1, \dots, p'_n\}$ with $p'_i = f(p_i)$ for $1 \leq i \leq n$, for each (c, m, k)-static clustering U' of P' the corresponding clustering U on P is a (c, m, k)-static clustering.*

Proof. Let t be a fixed but arbitrary point of time.

Proposition 4 *Let $V' = \{D'_1, \dots, D'_m\}$ be an arbitrary clustering of P' and let $V = \{D_1, \dots, D_m\}$ be the corresponding clustering of P. Let $d'_i[t]$ be an optimal cluster center for $D'_i[t]$ for $1 \leq i \leq n$. Then,*

1. *$d_i := f(d'_i)$ is an optimal cluster center for D_i for all i.*
2. *$\gamma(V, P, t) = t^2 \cdot \gamma(V', P', t)$*
3. *if V' is an optimal clustering of $P'[t]$, V is an optimal clustering of $P[t]$.*

Proof. Because of the properties of 1-means clustering, d'_i is the centroid of D'_i for all i, and therefore

$$\begin{aligned} d_i[t] = t \cdot d'_i[t] &= t \frac{1}{|D'_i|} \sum_{p'_j[t] \in D'_i[t]} p'_j[t] \\ &= t \frac{1}{|D'_i|} \sum_{p_j[t] \in D_i[t]} \frac{p_j[t]}{t} = \frac{1}{|D'_i|} \sum_{p_j[t] \in D_i[t]} p_j[t] \end{aligned}$$

is the centroid of $D_i[t]$ and of course an optimal cluster center for $D_i[t]$. Furthermore,

$$\begin{aligned} \gamma(V, P, t) &= \sum_{D_i \in V} \sum_{p_j \in D_i} \|p_j[t] - d_i[t]\|^2 \\ &= t^2 \cdot \sum_{D_i \in V} \sum_{p_j \in D_i} \sum_{1 \leq k \leq d} (p'_j[t]_k - d'_i[t]_k)^2, \end{aligned}$$

which equals $t^2 \cdot \gamma(V', P', t)$. This also implies that V is an optimal clustering for $P[t]$, if V' is optimal for $P'[t]$. □

Let U' be a (c, m, k)-static clustering of P' and let U be the corresponding clustering of P. We conclude

$$\begin{aligned} \gamma(U, P, t) = t^2 \cdot \gamma(U', P', t) &\leq t^2 \cdot c \cdot \gamma_{opt}(P', k, t) \\ &= t^2 \cdot c \cdot \left(\frac{1}{t^2} \cdot \gamma_{opt}(P, k, t) \right) = c \cdot \gamma_{opt}(P, k, t), \end{aligned}$$

using Proposition 4 and completing the proof. □

We will now use the preceeding lemmas for showing that one step of the recursion of the above algorithm maintains an approximation factor, if the stabbing guarantees an approximation and the recursive calls of the algorithm return approximate motion clusterings for the subproblems. One can easily use this to prove Theorem 5 by induction:

Theorem 5 *Let $P = \{p_1, \dots, p_n\}$ be a set of n polynomially moving points with a degree of motion of at most μ and let $\mathcal{A}$ be an algorithm that computes a (c, m, k)-stabbing for a given set of moving points, whereas $m \geq k$. Then one can compute a $(((1+\sqrt{c})^{\mu+1} - 1)^2, m^{\mu+1}, k)$-static clustering of P in running time $\mathcal{O}(n)$ plus the time for the calls of $\mathcal{A}$ and considering μ and the dimension d to be constant.*

Let us consider a step of the recursion. We are given a set of polynomially moving points $P = \{p_1, \dots, p_n\}$ with a degree of motion of at most $\mu \geq 1$. Now we compute an (a, m, k)-stabbing (C, t^*), whereas $C = \{C_1, \dots C_m\}$, on these points using an algorithm $\mathcal{A}$ as defined in Theorem 5. We move all point curves to the centroids of their corresponding stabbing clusters at the point of time defined by t^*, getting the point set C'_i for C_i for $i = 1, \dots, m$. Now, for all i, we divide all points in C'_i by their parameter t and get $\hat{C}_i$, on which we compute a (c, l, k)-static clustering $\hat{E}_i$ recursively for constants $a > 0$ and $l \geq k$. Because of Lemma 3, the corresponding clustering E'_i of C'_i is a (c, l, k)-static clustering.
Now consider the clustering $E' = \bigcup_{1 \leq i \leq m} E'_i$ of the moving point set $P' = \bigcup_{1 \leq i \leq m} C'_i$. We fix an arbitrary point of time t. The cost of E' at this point of time can be bounded by

$$\gamma(E', P', t) = \sum_{1 \leq i \leq m} \gamma(E'_i, C'_i, t) \leq c \sum_{1 \leq i \leq m} \gamma_{opt}(C'_i, k, t),$$

since all E'_i guarantee a c-approximate clustering for C'_i. Let $V' = \{V'_1, \dots, V'_k\}$ be an optimal k-Clustering for $P'[t]$. We intersect each cluster V'_j with every stabbing cluster $C'_i[t]$, getting a set of points V'_{ji}. Let us now consider the set $D'_i := \{V_{ji}\}_{1 \leq j \leq k}$. Clearly, D'_i is a k-clustering of $C'_i[t]$, and hence has at most the cost of the optimal k-clustering of $C'_i[t]$. Furthermore, all clusters of D'_i are subsets of clusters of V', so that the cost of each cluster V'_j of V' are at least the sum of the cost of the corresponding subclusters in $D_1, \dots, D_m$. We conclude

$$\gamma(E', P', t) \leq c \sum_{1 \leq i \leq m} \gamma(D'_i, C'_i, t) \leq c \cdot \gamma(V', P'[t]),$$

which equals $c \cdot \gamma_{opt}(P', k, t)$. Now we can apply Lemma 2 on E' and bound the cost of the corresponding clustering E of P, which has lm clusters, by $\gamma(E, P, t) \leq (\sqrt{c}(1+\sqrt{a})+\sqrt{a})^2 \cdot \gamma_{opt}(P, k, t)$.

For computing a k-means motion clustering it remains to show that one can compute a k-means stabbing which yields an $\mathcal{O}(1)$-approximation to the best clustering of any point of time for a moving point set P. Our algorithm computes an $(1+\epsilon)$-approximate k-stabbing in running time $\tilde{\mathcal{O}}\left(\frac{n^{1+k/\epsilon}}{(k-2)!}\right)$. For the sake of brevity, we will only outline this algorithm.
The algorithm gets a set P of n polynomially moving points, the number of clusters k and a proximity parameter ϵ as input. It enumerates all $\frac{n^{k/\epsilon}}{k!}$ possibilities for choosing (with replacement) k subsets of P, each having a size of $1/\epsilon$. Consider the point of time t that yields the minimum-cost k-clustering. It is known by [1] that one of the subset combinations we check yields an approximation to the optimal clustering at time t arbitrarily near to $(1+\epsilon)$.
Now we fix the k centroids of the considered subsets of P and assign the moving points in P to the nearest cluster center. Since for each point the optimal assignment can change at most $\lambda_{2\mu}(k) = \tilde{\mathcal{O}}(k)$ times, i.e. the maximum length of a Davenport-Schinzel Sequence with a degree of 2μ for k elements [1], we only have to consider $\tilde{\mathcal{O}}(nk)$ time intervals. Inside these intervals the assignment to the cluster centers is fixed, and we can compute the optimal point of time for each cluster by minimizing the sum of the distances of the points inside the cluster to the cluster center, which is a polynomial; we assume that, once we have computed all these polynomials in $\mathcal{O}(n)$, the minimization can be done in running time $\mathcal{O}(1)$ for each of them. We return the optimal stabbing of the best time interval of all possibilities to enumerate k subsets of size of $1/\epsilon$ of P, which, by the above discussion, is an $(1+\epsilon, k, k)$-stabbing of P.
Combining this with theorem 5, we conclude:

Theorem 6 *For a set $P \subseteq \mathbb{R}^d$ of n polynomially moving points with a degree of motion of at most μ, a proximity parameter $\epsilon > 0$ and $k \in \mathbb{N}$, one can compute a $((1+\sqrt{1+\epsilon})^{\mu+1}-1)^2, k^{\mu+1}, k)$-static clustering in running time $\tilde{\mathcal{O}}\left(\frac{n^{k/\epsilon+1}}{(k-2)!}\right)$, whereas d and μ are considered to be constants.*

Coresets for k-Means Motion Clustering In this Paragraph we outline the construction of coresets for k-means motion clustering. Let us at first define coresets for static point sets.
A weighted set of moving points is a set Q of moving points in $\mathbb{R}^d$ together with a weight function $w : Q \to \mathbb{R}_0^+$. When calculating the cost of a clustering C of such a point set at a point of time t, we multiply the distance of each point to its cluster center by its weight, i.e. $\gamma_w(C,P) = \sum_{C_i \in C} \sum_{p \in C_i} w(p) \cdot \|p - \hat{c}_i\|^2$. Let T be a weighted set of moving points and P be a set of moving points. At an arbitrary point of time t, we call $T[t]$ a (k,ϵ)-coreset of $P[t]$, if for all sets $D \in \mathbb{R}^d$ of k points and the clustering C on $P[t]$ and C' on $T[t]$ that are defined by D by assigning each point to the nearest point in D, it holds $(1-\epsilon)\gamma(C,P,t) \leq \gamma_w(C',T,t) \leq (1-\epsilon)\gamma(C,P,t)$. We call T a static (k,ϵ)-coreset of P, if it is a (k,ϵ)-coreset for all points of time simultanously.
Our result is the following theorem. It is based on a coreset result for static point sets of Har-Peled and Mazumdar [2]; we ommit the proof for the sake of brevity.

Theorem 7 *Let P be a set of n polynomially moving points with a degree of motion bounded by μ and let $\mathcal{A}$ be an algorithm that retrieves a (c, am, m)-stabbing for any such set of points and for all m, for constants $a \in \mathbb{N}$ and $c \geq 1$. Then, one can compute a static (k,ϵ)-coreset T of size $\mathcal{O}\left(\frac{(k \log n)^{\mu+1}}{\epsilon^d}\right)$ of P, whereas any point in T is a polynomial with a degree of motion of at most μ. The running time is $\mathcal{O}(n)$ plus the time needed for the execution of $\mathcal{A}$.*

References

[1] P. Agarwal, M. Sharir: Davenport-Schinzel Sequences and Their Geometric Applications; Cambridge University Press, 1995

[2] S. Arora. P. Raghavan, S. Rao: Approximation Schemes for Euclidean k-Medians and Related Problems; 30th annual ACM Symposium on Theory of Computing, 1998

[3] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick und M. Segal: Mobile facility location; Proc. of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, p. 46, 2000

[4] A. Czumaj, G. Frahling, C. Sohler: Efficient kinetic data structures for MaxCut. In: Proceedings of the 19th Canadian Conference on Computional Geometry (CCCG), p. 157, 2007

[5] B. Degener, J. Gehweiler, C. Lammersen: Kinetic Facility Location, Algorithmica, special issue on selected papers from SWAT'08, 2008

[6] F. Hellweg: Clustering unter Bewegung für k-Means- und k-Median-Zielfunktionen, Master Thesis, 2008

[7] J. Gao, L. Guibas, J. Hershberger, L. Zhang, A. Zhu: Discrete mobile centers. Discrete Comput. Geom. 30(1), p. 45, 2003

[8] J. Gao, L. Guibas, A. Nguyen: Deformable spanners and applications. Comput. Geom. 35(12), p. 2, 2006

[9] S. Har-Peled: Clustering Motion, Discrete & Comput. Geom. 31(4), 2004

[10] J. Hershberger: Smooth kinetic maintenance of clusters. Comput. Geom. Theory Appl. 31(12), p. 45, 2005

[11] M. Inaba, N. Katoh, H. Imai: Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering (ext. abstract); Proc. of the 10th annual Symposium on Comp. Geom., 1994

The Maximum Box Problem for Moving Points on the Plane

S. Bereg [*] J.M. Díaz-Báñez [†] P. Pérez-Lantero [‡] I. Ventura [§]

Abstract

Given a set R of r red points and a set B of b blue points on the plane, the static version of the Maximum Box Problem is to find an isothetic box H such that $H \cap R = \emptyset$ and the cardinality of $H \cap B$ is maximized. In this paper, we consider a kinetic version of the problem where the points in $R \cup B$ move according to algebraic functions. We design a compact and local quadratic-space kinetic data structure (KDS) for maintaining the optimal solution in $O(r \log r + r \log b)$ time per each event. We also consider a more general static problem where the maximum box can be arbitrarily oriented. This is an open problem in [1]. We show that our approach can be used to solve this problem in $O((r+b)^2(r \log r + r \log b))$ time.

1 Introduction

In Pattern Recognition and Classification problems, a natural method for selecting prototypes that represent a class is to perform cluster analysis on the training data [6]. Typically, two sets of points X^+ and X^- are given, and one would like to find patterns which intersect exactly one of these sets. The clustering can be obtained by using simple geometric shapes such as circles or boxes. Recent papers deal with the *Maximum Box Problem*, where the clustering is due by considering maximum boxes, i.e., boxes containing the maximum number of points in the given data set. See [8, 9]. The basic problem is the following: given a finite set of blue points $B = X^+$ and a finite set of red points $R = X^-$ on the plane, compute a box (axis-aligned rectangle) containing the maximum number of points of B but avoiding points of R. In many applications, the data are given in a dynamic scenario. For instance, in fixed wireless telephony access and driving assistance, detection and recognition of patterns for moving objects are key functions [2]. In fact, with the continued proliferation of wireless communication and advances in positioning technologies, algorithms to efficiently solve optimization problems about large populations of moving data are gaining interest. New devices offer to the companies an opportunity of providing a diverse range of e-services, many of which will exploit knowledge of the users changing location. This results in new challenges to database and classification technology [3].

In this paper, we introduce and investigate the dynamic version of the Maximum Box Problem where the dataset is modeled by points moving according to specified algebraic functions. We present a *Kinetic Data Structure* (KDS) to efficiently maintain the maximum box for a bi-colored set of moving points. A KDS is used for keeping track the attributes of interest in a system of moving objects [7]. The plan of the paper is as follows: In section 2 we give an algorithm for the static case that will be useful for the dynamic version. A new data structure for maintaining the maximum box in a kinetic setting is proposed in section 3. Finally, in section 4 we give an algorithm for the Maximum Box Problem in a static setting for which the box can be arbitrarily oriented.

2 The Static Version

The static version of the maximum box problem in the plane has been already studied. In [8], an $O(b^2 \log b + br + r \log r)$-time algorithm has been presented, where $r = |R|$ and $b = |B|$. We propose here a similar algorithm which is used in the dynamic case for the designment of our data structure. First, we observe that a maximum box for B and R can be enlarged until each of its sides either touches a red point or reaches the infinity. Thus, the maximum box can be transformed to one of the following isothetic objects with red points on its boundary and no red points inside: a rectangle, a half-strip, a strip, a quadrant or a half-plane (see Fig. 1).

We show how to compute the maximum box of type 1), 2) or 3). The cases of a quadrant 4) and a half-plane 5) are even easier and can be addressed by applying the techniques based on the *Dynamic Bentley's Maximum Subsequence Sum Problem* that were presented in [5]. Both cases can be solved in time $O(n \log n)$.

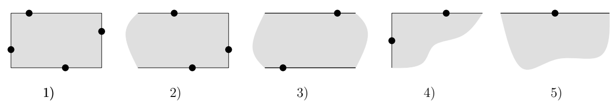


Figure 1: Configurations of the maximum box.

*University of Texas, besp@utdallas.edu

†Universidad de Sevilla, dbanez@us.es, partially supported by grant FEDER-MTM2006-03909.

‡Universidad de la Habana, pablo@matcom.uh.cu, supported by MAEC-AECI.

§Universidad de Sevilla, iventura@us.es, partially supported by grant FEDER-MTM2006-03909.

We proceed as follows. For every red point p_r, we compute a rectangle $H(p_r)$ such that: (i) the top side contains p_r, (ii) the interior contains the maximum number of blue points, and (iii) the boundary has only red points. Then, we take the best of all $H(p_r)$. To do this for each red point p_r, we draw a horizontal line l passing through p_r and sweep the points below l by moving l downwards.

Let h_r be the horizontal line that passes through p_r. During the sweeping we maintain two balanced binary search trees T_R and T_B such that T_R (resp. T_B) contains all the red (resp. blue) points that lie between h_r and l sorted by x-coordinate. In the tree T_B, for each node v we also maintain a counter of the number of blue points on the subtree rooted at v. When l passes through a point p we do the following. Let $left(p)$ (resp. $right(p)$) be the rightmost (resp. leftmost) red point in T_R located to the left (resp. right) of the vertical line that passes through p, and let x_{left} (resp. x_{right}) be its x coordinate. If $left(p)$ (resp. $right(p)$) does not exist then x_{left} (resp. x_{right}) is $-\infty$ (resp. $+\infty$). We only process p if the x coordinate of p_r lies in the interval (x_{left}, x_{right}). In that case, p is inserted in T_R or T_B according to its color and if p is a red point, we consider a candidate $H(p_r)$ as the isothetic rectangle (half-strip or strip) whose sides pass through $left(p)$, $right(p)$, p_r and p. The count of blue points inside $H(p_r)$ is equal to the count of blue points in T_B whose x coordinate lies in (x_{left}, x_{right}).

In the above procedure $left(p)$, $right(p)$ and the count of blue points inside $H(p_r)$ are obtained in logarithmic time each, thus the top-down sweeping from p_r is done in $O(r \log r + r \log b + b \log b)$ time, giving then an overall $O(r^2 \log r + r^2 \log b + rb \log b)$-time and $O(r + b)$-space process. Within the same time we can find the smallest axis-parallel rectangle (box) that maximizes the number of covered points from B and does not contain any point from R, improving the $O(n^3 \log^4 n)$-time algorithm given by [9].

3 The Maximum Box Problem for moving points

In this section we introduce a new kinetic data structure (KDS), for maintaining the maximum box. A KDS is a structure that maintains an attribute of a set of continuously moving objects [7]. It consists of two parts: a combinatorial description of the attribute and a set of certificates. The certificates are elementary tests on the input objects with the property that as long as their outcomes do not change, the attribute does not.

Lemma 1 *Let X (resp. Y) be the elements of $R \cup B$ sorted by abscissa (resp. ordinate). The maximum box for R and B is univocally determined by X and Y, and it does not change combinatorially over time as long as X and Y do not.*

Our KDS is named as *Maximum Box Kinetic Data Structure* (MBKDS) and we will proceed now with its ingredients. In our problem, the set of moving objects is $R \cup B$ and the attribute is its maximum box. According to the previous lemma, we consider as a certificate the condition that two consecutive points in X (resp. Y) satisfy the x-order (resp. y-order). An event is the failure of a certificate at some instant of time t and it implies an update of the MBKDS. We denote each event as a pair $\langle c, t \rangle$ where c is the certificate and t is the instant of time where c is violated. We name an event as *flip*.

A KDS is *compact* if it has *few* certificates and *local* if no object participates in *too many* certificates. With this it can be updated easily when the flight plan of an object changes (see [7]). It is easy to prove that the MBKDS is compact and local.

The MBKDS is composed by the event queue $\mathcal{E}$ and the structure $\mathcal{D} = \{\mathcal{D}(p_r) : p_r \in R\}$ where $\mathcal{D}(p_r)$ is a data structure for each red point p_r. When an event corresponding to the certificate of two consecutive elements of X, say $[X_i, X_{i+1}]$, occurs it is removed from $\mathcal{E}$. Then, we remove from $\mathcal{E}$ the events corresponding to $[X_{i-1}, X_i]$ and $[X_{i+1}, X_{i+2}]$. After that, we swap X_i and X_{i+1} and insert in $\mathcal{E}$ new events for $[X_{i-1}, X_i]$, $[X_i, X_{i+1}]$ and $[X_{i+1}, X_{i+2}]$. The event time is computed based on our knowledge of the motions of X_{i-1}, X_i, X_{i+1} and X_{i+2}. $\mathcal{E}$ is designed as a priority queue with its basic operations in $O(\log n)$ time. The certificates $[Y_i, Y_{i+1}]$ are treated similarly.

The definition and details of $\mathcal{D}(p_r)$ are as follows. Consider first the horizontal line h_r passing through the red point p_r and let $Red(h_r)$ be the set of red points lying below h_r (see Fig. 2). Denote as $x(p)$ (resp. $y(p)$) the abscissa (resp. ordinate) of p. For each $p \in Red(h_r)$ we define:

- $\mathcal{V}(p)$ as the vertical half-line for which p is its top-most point.
- $\mathcal{I}(p)$ as the maximum-length horizontal segment that contains p and does not intersect any other $\mathcal{V}(q)$ for q in $Red(h_r) \setminus \{p\}$.
- $Blue(s)$ as the set of blue points that lie on the rectangle whose bottom side is the horizontal segment s and its top side lies on h_r.
- $left(p)$ (resp. $right(p)$) as the rightmost (resp. leftmost) point in $Red(h_r)$ located to the left (resp. right) of the vertical line that passes through p and whose y coordinate is greater than $y(p)$ (see Fig. 2). Notice that the end points of $\mathcal{I}(p)$ lie on $\mathcal{V}(left(p))$ and $\mathcal{V}(right(p))$.
- the set $candidates(p_r)$ of the points $p \in Red(h_r)$ such that $\mathcal{I}(p)$ intersects the vertical line that passes through p_r. Observe that, according to the

previous section, $H(p_r)$ is the box whose top side lies on h_r and its bottom side is the interval $\mathcal{I}(p)$ such that $p \in \mathit{candidates}(p_r)$ and $|\mathit{Blue}(\mathcal{I}(p))|$ is maximized.

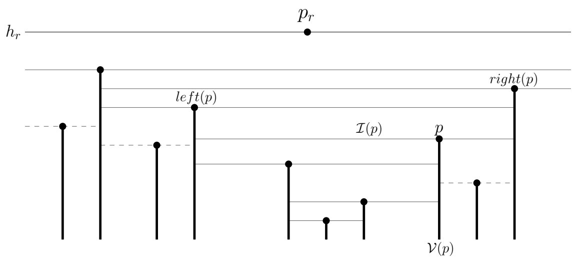


Figure 2: $\mathcal{D}(p_r)$. Candidate points are those whose $\mathcal{I}(p)$ is drawn with a continuous line.

The key idea for $D(p_r)$ is to dynamically compute $H(p_r)$ when two points that lie below h_r make a flip. $\mathcal{D}(p_r)$ is designed as follows:

- The elements of $\mathit{candidates}(p_r)$ are stored in the leaves of a dynamic balanced binary search tree $\mathcal{Q}$ in decreasing order of ordinate. Every node v is labeled with a such that $|\mathit{Blue}(\mathcal{I}(p))|$ is equal to the sum of the a labels of the nodes in the unique simple path from the leaf that represents p to the root. In addition, every internal node v is labeled with b whose value is the candidate p in the subtree rooted at v that maximizes $|\mathit{Blue}(\mathcal{I}(p))|$. With this, the label b of the root is the candidate red point that determines $H(p_r)$.
- A balanced binary search tree $\mathcal{T}_R$ whose elements are the elements of $\mathit{Red}(h_r)$ in increasing order of abscissa. Every node v is labeled with y_{max} that is the element that has the maximum ordinate on the subtree rooted at v. It permits us to obtain in logarithmic time $left(p)$ and $right(p)$ for any $p \in \mathit{Red}(h_r)$.
- For each $p \in \mathit{Red}(h_r)$ we divide $\mathcal{I}(p)$ at p obtaining the horizontal segments $\mathcal{I}_{left}(p)$ and $\mathcal{I}_{right}(p)$ (see Fig. 3). Let $\mathcal{T}_{left}(p) = \{\mathcal{I}_{right}(q) : q \in \mathit{Red}(h_r) \setminus \{p\} \wedge right(q) = p\}$ and $\mathcal{T}_{right}(p) = \{\mathcal{I}_{left}(q) : q \in \mathit{Red}(h_r) \setminus \{p\} \wedge left(q) = p\}$. The set $\mathcal{T}_{left}(p)$ is represented in a dynamic balanced binary search tree where its elements $\mathcal{I}_{right}(q)$ are stored at the leaves in decreasing order of $y(q)$. Every node v is labeled with a in a similar way as we did for $\mathcal{Q}$. In this case $|\mathit{Blue}(\mathcal{I}_{right}(q))|$ is the sum of the a labels of the nodes in the path from the leaf that represents $\mathcal{I}_{right}(q)$ to the root. Also, every node v is labeled with the boolean c indicating if all the elements $\mathcal{I}_{right}(q)$ in the subtree rooted at v satisfy that q is a candidate red point. $\mathcal{T}_{left}(p)$ is designed to support the operators *join* and *split* both in logarithmic time [4].

  The set $\mathcal{T}_{right}(p)$ is represented in the same manner.
- The set $\{\mathcal{I}(p) : p \in \mathit{Red}(h_r)\} \cup \{\mathcal{V}(p) : p \in \mathit{Red}(h_r)\}$ determines a partition $\mathcal{P}$ of the lower half-plane of h_r. For each cell $\mathcal{C}$ of $\mathcal{P}$ we store in a balanced binary search tree $\mathcal{T}_B(\mathcal{C})$ (supporting operators *join* and *split*) the elements of $B \cap \mathcal{C}$ in decreasing order of ordinate. Each node v is labeled with the count of blue points on the subtree rooted at v in such a way the label of the root is $|B \cap \mathcal{C}|$. Each blue point p below h_r has a reference $\mathcal{T}_{\mathcal{C}}(p)$ to the tree $\mathcal{T}_B(\mathcal{C})$ where $\mathcal{C}$ is the cell that contains p. Also, we associate to each $p \in \mathit{Red}(h_r)$ the tree $\mathcal{T}_B(p)$, which is the tree of blue points that corresponds to the cell of $\mathcal{P}$ that is bounded below by $\mathcal{I}(p)$, and the tree $\mathcal{T}_{B_{right}}(p)$ that corresponds to the cell of $\mathcal{P}$ bounded by $\mathcal{V}(p)$ and $\mathcal{V}(q)$ and has no bottom boundary, where q is the element that follows p on $\mathcal{T}_R$. (see Fig. 3).

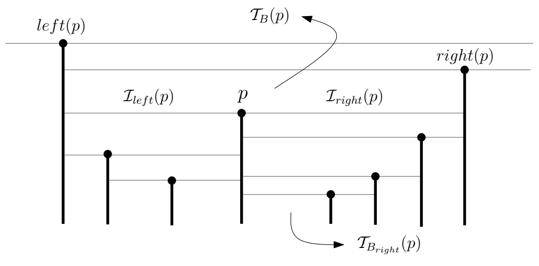


Figure 3: Details of $\mathcal{D}(p_r)$.

Because of the similarity between the structure of $\mathcal{D}(p_r)$ and the operations of the algorithm of section 2, a similar algorithm to build $\mathcal{D}(p_r)$ can be designed. We establish the following result,

Theorem 2 *Given R, B and p_r such that $|R| = r$, $|B| = b$ and $p_r \in R$, $\mathcal{D}(p_r)$ requires $O(r+b)$ space and it can be built in $O(r\log r + r\log b + b\log b)$ time.*

Corollary 3 *Given R and B such that $|R| = r$ and $|B| = b$, the data structure $\mathcal{D}$ has space complexity $O(r^2 + rb)$ and it can be built in $O(r^2\log r + r^2\log b + rb\log b)$ time.*

In the following we show all the possible types of flips that can occur when the elements of $R \cup B$ move. The details on how to process any of them depend on its type and are given in the full version. We say that a flip is *horizontal* (resp. *vertical*) when the two involved points change their y-order (resp. x-order).

Flip type (A): Two blue points make a vertical flip. Nothing to do.

Flip type (B): Two blue points b_1 and b_2 make a horizontal flip. Exchange b_1 and b_2 in every tree of blue points $\mathcal{T}_B(.)$ or $\mathcal{T}_{B_{right}}(.)$ that contains both.

Flip type (C): A red point r_1 and a blue point b_2 make a horizontal flip. Consider five cases as depicted in Figure 4.

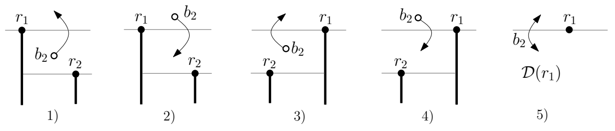


Figure 4: Horizontal flip cases between a red point r_1 and a blue point b_2.

Flip type (D): A red point r_1 and a blue point b_2 make a vertical flip. Consider two cases as depicted in Figure 5, and their symmetric cases.

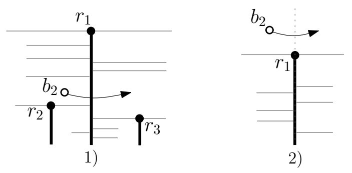


Figure 5: Vertical flip cases between a red point r_1 and a blue point b_2.

Flip type (E): Two red points r_1 and r_2 make any flip: we have to consider two cases as depicted in Figure 6, and their symmetric cases. When the flip is vertical the operators *join* and *split* [4] are useful.

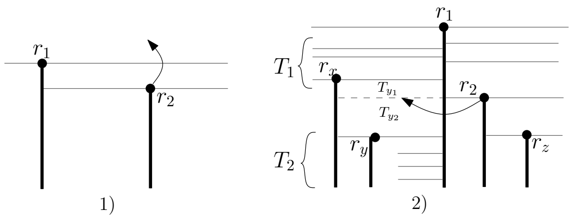


Figure 6: Flip cases between two red points r_1 and r_2. 1) Horizontal 2) Vertical.

Theorem 4 *Given a set of moving red points R and a set of moving blue points B such that $|R| = r$ and $|B| = b$, whenever two points in $R \cup B$ make a flip, the data structure MBKDS can be updated in $O(r \log r + r \log b)$ time.*

4 The Arbitrarily Oriented Maximum Box Problem

In this section we consider, as an application of our method for the dynamic case, the problem of finding the Maximum Box for a static set $R \cup B$ on the plane, when the box can be oriented according with any angle (direction) in $[0, \pi]$.

It is easy to see that there are $O((r + b)^2)$ critical directions and the method of section 2 can be applied for each of them. With this we obtain an $O((r+b)^2(r^2 \log r + r^2 \log b + rb \log b))$-time algorithm. However, we can do it better by iterating all the critical directions and computing dynamically the Maximum Box per each. Start with the direction given by the angle $\theta = 0$. Compute the isothetic Maximum Box and the data structure $\mathcal{D}$. Then make a rotational sweeping of the coordinate axis maintaining the x-order and the y-order of the elements of $R \cup B$. The x-order (resp. y-order) changes whenever two points are at the same distance to the y-axis (resp. x-axis), implying the swap (*flip*) of two consecutive elements and the appearance of a new critical orientation θ. Thus, the Maximum Box that is oriented according to θ is computed by making the $O(r \log r + r \log b)$-time update in $\mathcal{D}$. The overall rotation angle is at most π radians and we establish the following result.

Theorem 5 *Given a set of red points R and a set of blue points B on the plane such that $|R| = r$ and $|B| = b$, the Arbitrarily Oriented Maximum Box Problem can be solved in $O((r + b)^2(r \log r + r \log b))$ time and $O(r^2 + rb)$ space.*

References

[1] B. Aronov and S. Har-Peled. On approximating the depth and related problems. SIAM J. Comput. Vol. 38, 899–921, 2008.

[2] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. In Proc. of the Second International Conf. on Mobile Data Management, Hong Kong. Lecture Notes Comp. Sci., vol. 1987, 3 – 14, 2001.

[3] A. Civilis, C. S. Jensen, S. Pakalnis. Techniques for Efficient Road-Network-Based Tracking of Moving Objects. IEEE Transactions On Knowledge And Data Engineering, Vol. 17, No. 5, 698–712, 2005.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*, Second Edition. MIT Press, McGraw-Hill, 2001.

[5] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, I. Ventura. Bichromatic separability with two boxes: a general approach. To appear in Journal of Algorithms.

[6] R. Duda, P. Hart, D. Stork. *Pattern Classification. John Wiley and Sons, Inc.*, New York, 2001.

[7] L.J. Guibas. Kinetic Data Structures: A State of the Art Report. In *Robotics: The Algorithmic Perspective.* A. K. Peters, Ltd 191–209, 1998.

[8] Y. Liu, M. Nediak. Planar Case of the Maximum Box and Related Problems. In Proc. Canad. Conf. Comp. Geom., Halifax, Nova Scotia, August 11–13, 2003.

[9] M. Segal. Planar Maximum Box Problem. Journal of Mathematical Modeling and Algortihms. 3, 31–38, 2004.

Minimizing the weighted directed Hausdorff distance between colored point sets under translations and rigid motions*

Christian Knauer † Klaus Kriegel † Fabian Stehn †

Abstract

Matching geometric objects with respect to their Hausdorff distance is a well investigated problem in Computational Geometry with various application areas. The variant investigated in this paper is motivated by the problem of determining a matching (in this context also called *registration*) for neurosurgical operations. The task is, given a sequence $\mathcal{P}$ of weighted point sets (anatomic landmarks measured from a patient) and a second sequence $\mathcal{Q}$ of corresponding point sets (defined in a 3D model of the patient), to compute the transformation that minimizes the *weighted* directed Hausdorff distance of $\mathcal{P}$ to $\mathcal{Q}$. The weighted Hausdorff distance, as introduced in this paper, takes the weights of the point sets into account, which in this application reflect the precision with which landmarks can be measured.

We present an exact solution for translations in the plane, a simple 2-approximation as well as a FPTAS for translations in arbitrary dimension and a constant factor approximation for rigid motions in the plane or in $\mathbb{R}^3$, the relevant cases for the mentioned application.

1 Introduction

A central aspect in pattern recognition and matching applications is the measure which is used to compare two objects. It defines which features of the objects are relevant for deciding about their similarity. An established measure to compare two geometric objects A and B is the (directed) Hausdorff distance $h(A, B)$, defined as $h(A, B) := \max_{a\in A} \min_{b\in B} \|a - b\|$, where $\|\cdot\|$ denotes the Euclidean norm.

We present an extension of the directed Hausdorff distance which is motivated by a registration application for neurosurgical operations. The task is to compute the transformation that matches a patient in an operation theatre to a 3D model of the patient, generated from CT/MRT scans. A matching is computed based on point sets, so-called anatomic landmarks, that are defined by the surgeon in the model during the operation planning phase, and measurements of these landmarks (by touching the appropriate spots of the patient with a traceable device) in the first phase of the operation. Landmarks from different anatomic regions can be measured with different precision. The differences in the accuracy are taken into account by assigning corresponding weights to each landmark. The magnitude of a weight that is assigned to an anatomic landmark is based on an empirical study about the precision of measuring landmarks, which will be published elsewhere.

Various distance measures based on the Hausdorff distance have been introduced and extensively studied by the Computer Graphics community. Standard applications in this line of research are comparing pixel images, e.g., for detecting [5] or recognizing faces in them.

1.1 Problem Definition

Given two sequences of finite point sets $\mathcal{P} = (P_1, \ldots, P_k)$ and $\mathcal{Q} = (Q_1, \ldots, Q_k)$ with $P_i, Q_i \subset \mathbb{R}^d$ and a sequence of weights $w = (w_1, \ldots, w_k)$ with $w_i \in \mathbb{R}^+$ and $w_1 \geq w_2 \geq \cdots \geq w_k$. For $n_i := |P_i|$ and $m_i := |Q_i|$ let $n = \sum_{i=1}^k n_i$ and $m = \sum_{i=1}^k m_i$. For the purpose of illustration one can think of $\mathcal{P}$ and $\mathcal{Q}$ being colored in k different colors, where all points in P_i and Q_i are colored with the i^{th} color.

Definition 1 *The* weighted directed Hausdorff distance $h(\mathcal{P}, \mathcal{Q}, w)$ *for* $\mathcal{P}$, $\mathcal{Q}$ *and* w *is defined as:*

$$h(\mathcal{P}, \mathcal{Q}, w) = \max_{1\leq i\leq k} w_i \; h(P_i, Q_i).$$

For points in the plane, $h(\mathcal{P}, \mathcal{Q}, w)$ can be computed in $O((n + m) \log m)$ time using Voronoi diagrams to compute the directed Hausdorff distance for every color; for higher dimensions it trivially can be computed in $O(mn)$ time.

In this paper we discuss the problem of matching two sequences of weighted point sets under the weighted Hausdorff distance. Formally, given $\mathcal{P}$, $\mathcal{Q}$ and w as before and a transformation class $\mathcal{T}$, find the transformations $t \in \mathcal{T}$ that minimize the function

$$f(t) = h(t(\mathcal{P}), \mathcal{Q}, w), \qquad (1)$$

where $t(\mathcal{P}) := (\{t(p) \mid p \in P_i\})_{1\leq i\leq k}$. For the remainder of this paper, let μ_{opt} denote the minimum of f.

*supported by the German Research Foundation (DFG), grant KN 591/2-2

†Institut für Informatik, Freie Universität Berlin, {knauer|kriegel|stehn}@inf.fu-berlin.de

In section two we present a 2-approximation for colored point sets in arbitrary dimension under translations, which is extended to a FPTAS in section three. In section four a constant-factor approximation for rigid motions is presented. An exact algorithm optimizing the Hausdorff distance under translations for points in the plane is presented in section five.

2 A 2-Approximation for Translations in $\mathbb{R}^d$

Theorem 1 *A 2-approximation for translations can be computed in $O(m_1(m+n)\log m)$ time in 2-space and in $O(m_1 mn)$ time for point sets in higher dimensions.*

Proof. Choose a point $p \in P_1$ and let T be the set of all translation vectors that move p upon a point of Q_1:

$$T := \{q - p \mid q \in Q_1\}.$$

Then, the translations t_{2app} of T that realize the smallest Hausdorff distance are 2-approximations of μ_{opt}:

$$f(t_{2app}) = \min_{t\in T} h(t(\mathcal{P}), \mathcal{Q}, w) \leq 2\mu_{opt}.$$

To show this, assume $\mathcal{P}$ to be in optimal position and let $\mathrm{nn}(p', Q_i) \subseteq Q_i$ be the set of nearest neighbors of $p' \in P_i$ in Q_i and let $n_i(p')$ be one representative of this set. In optimal position, all weighted distances are bounded by μ_{opt}:

$$\forall 1 \leq i \leq k \; \forall p' \in P_i \quad w_i\|p' - n_i(p')\| \leq \mu_{opt},$$

where $\|\cdot\|$ denotes the Euclidean norm. Consider the translation $t_q \in T$ that moves the chosen point p upon one of its nearest neighbors $q \in \mathrm{nn}(p, Q_1)$. By applying t_q to $\mathcal{P}$, all points in any P_i are moved away from their optimal position by $\|p-q\|$. The new weighted distance l of any $\tilde{p} \in P_i$ to one of its closest neighbors after applying t_q is at most:

$$\begin{aligned} l &= w_i\; h(t_q(\tilde{p}), Q_i) \\ &\leq w_i\; (\|\tilde{p} - n_i(\tilde{p})\| + \|p - n_1(p)\|) \\ &\leq \mu_{opt} + w_i\; \|p - n_1(p)\| \\ &\leq \mu_{opt} + w_1\; \|p - n_1(p)\| \\ &\leq 2\mu_{opt}. \end{aligned}$$

For a fixed $p \in P_1$ all translations that move p upon a point $q \in Q_1$ have to be tested, which yields the stated runtime. □

The key argument that guarantees an approximation factor of 2 is that w_1 is at least as large as any other weight. Choosing a $p' \in P_i$ for any color i and testing all translation vectors that move p' onto a point $q' \in Q_i$ yields in a $1 + {}^{w_1}\!/_{w_i}$-approximation that can be computed in $O(m_i(m+n)\log m)$ respectively $O(m_i\, m\, n)$ time. Together with the the pigeonhole principle, this implies that there is a $1 + ({}^{w_1}\!/_{w_k})$-approximation that can be computed in $O\left({}^{m^2}\!/_k\, n\right)$ time for general d and in $O\left({}^{m}\!/_k\,(m+n)\log m\right)$ time in the plane.

3 A FPTAS

Theorem 2 *For every $\epsilon > 0$ a $(1+\epsilon)$-approximation for translations can be computed in $O\left({}^{1}\!/_{\epsilon^2}\, m_1(m+n)\log m\right)$ time in 2-space and in $O\left(\left({}^{\sqrt{d}}\!/_{\epsilon}\right)^d m_1\, m\, n\right)$ time for higher dimensions.*

Proof. Let $\mu_{2app} = f(t_{2app})$ where t_{2app} is a solution of the 2-approximation as described in Section 2. For the given ϵ, choose any $p \in P_1$ and for every $q \in Q_1$ build a regular squared grid G_q^ϵ centered in q with side length μ_{2app}/w_1 where each cell has a side length of $(\epsilon\,\mu_{2app})/(\sqrt{d}\,w_1)$. Let T be the set of all translations, that move a $p \in P_1$ onto a grid point of a grid G_q^ϵ for any $q \in Q_1$. The translations t_{app} of T that realize the smallest weighted Hausdorff distance satisfy

$$f(t_{app}) = \min_{t\in T} h(t(\mathcal{P}), \mathcal{Q}, w) \leq (1+\epsilon)\mu_{opt}.$$

Assume $\mathcal{P}$ to be in optimal position and let $q = n_1(p)$ be one of the nearest neighbors of the chosen $p \in P_1$. By construction, the point p lies within the boundaries of the grid G_q^ϵ. Due the choice of the cell length, the furthest distance l of p to one of its closest grid points is at most $l \leq (\epsilon\,\mu_{2app})/(2\sqrt{d}w_1)$ and as $\mu_{opt} \geq \mu_{2app}/2$ we have that $l \leq (\epsilon\,\mu_{opt})/(\sqrt{d}w_1)$. As all w_i for $1 < i \leq k$ are at most as large as w_1, the weighted distance of all other points increases by at most $\epsilon \cdot \mu_{opt}$ when translated along the distance l. There are $\left(\frac{\sqrt{d}}{\epsilon}\right)^d$ grid points for each G_q^ϵ that need to be tested, which implies the stated runtime. □

Using the same arguments as for the 2-approximation, its easy to show that a $(1+\epsilon)$ approximation can be computed in $O\left(\left({}^{\sqrt{d}\,c}\!/_{\epsilon}\right)^d m^2\; n/k\right)$ time for point sets in dimension $d \geq 3$ or in $O\left(\left({}^{c}\!/_{\epsilon}\right)^2\; m/k(m+n)\log m\right)$ time in 2-space with $c = {}^{w_1}\!/_{w_k}$.

4 A Constant Factor Approximation for Rigid Motions in the Plane

Due to limited space, we briefly describe how to extend the results for the constant factor approximation for translations in Section 2 to the transformation class of rigid motions, i.e., translations plus rotations.

Theorem 3 *A $2(1+\sqrt{2})$-approximation for rigid motions can be computed in $O(m_1\, m_i\,(m+n)\log m)$ time for point sets in the plane.*

Sketch of the algorithm. Choose any $p \in P_1$. Let $P := \bigcup_{1\leq i\leq k} P_i$ and let $\omega : P \to \mathbb{R}$ be the function that maps a point $x \in P$ to its weight $\omega(x) = w_i$ for $x \in P_i$. Let p' be a point that maximizes the weighted distance to p:

$$p' \in \arg\max_{x\in P} \omega(x)\|x - p\|,$$

w.l.o.g. let $\omega(p') = w_i$. This choice of p' ensures that the change of the Hausdorff distance of any point $\tilde{p} \in P$ when rotated around p is bounded by the change of the weighted Hausdorff distance of p'.
For any $q \in Q_1$ and any $q' \in Q_i$ let t be defined as $t := t_{q,q'} \circ t_q$, where t_q is the translation that moves p upon q and $t_{q,q'}$ is the rotation around q by the smallest angle such that q, q' and $t_{q,q'} \circ t_q(p')$ are aligned, see Figure 1.

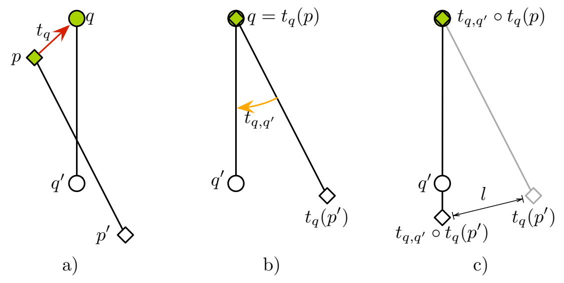


Figure 1: **a)** The translational part t_q, **b)** the rotational part $t_{q,q'}$, **c)** the aligned configuration.

For $p \in P_1$ and $p' \in P_i$ as chosen before let T be the set of all transformations that align $p \in P_1$ and $p' \in P_i$ with some $q \in Q_1$ and $q' \in Q_i$ in the just described manner.

The transformations t_{rapp} of T that realize the smallest Hausdorff distance satisfy the stated approximation factor:

$$f(t_{rapp}) = \min_{t\in T} h(t(\mathcal{P}), \mathcal{Q}, w) \leq 2(1+\sqrt{2})\mu_{opt}.$$

4.1 Extending the Result to $\mathbb{R}^3$

In $\mathbb{R}^3$ one additional degree of freedom has to be fixed. Based on the registration in the previous section, this freedom is a rotation around the axis $\overline{pp'}$. Choose p and p' as before and let $p'' \in P_j$ be a point that maximizes the weighted distance to the line through p and p'. For any $q'' \in Q_j$ consider additionally the rotation $t_{q,q',q''}$ around the axis qq' such that $t_{q,q',q''} \circ t_{q,q'} \circ t_q(p'')$, q, q' and q''' are coplanar. It is easy to see, that the additional rotation results in a $6 + 2\sqrt{2}$-approximation of the optimum and the runtime increases to $O(m_1\, m_i\, m_j\, mn)$.

5 An Exact Algorithm for Computing a Registration in the Plane under Translations

The minima of the objective function $f(t)$ (Equation 1) under translations in the plane can be computed similar to the approach from Huttenlocher et. al. [3] by partitioning the translation space into cells such that for all translations of one cell the weighted Hausdorff distance is realized by the same point pair.

Recall, that the function f is given as:

$$\begin{aligned} f(t) &:= \max_{1\leq i\leq k} w_i\, h(t(P_i), Q_i) \\ &= \max_{1\leq i\leq k} w_i \max_{p\in P_i} \min_{q\in Q_i} \|(p+t) - q\| \\ &= \max_{1\leq i\leq k} w_i \max_{p\in P_i} \min_{q\in Q_i} \|(q-p) - t)\| \end{aligned}$$

So, computing the distance of the point $p + t$ to q is equivalent to computing the distance of the translation vector t to the point $q - p$. By introducing the sets $S_i(p) := \{q - p | q \in Q_i\}$ we can formulate $f(t)$ as

$$f(t) = \max_{1\leq i\leq k} w_i \max_{p\in P_i} \min_{c\in S_i(p)} \|c - t\|.$$

To compute the minima of f, we compute the decomposition of the translation space into (potentially empty) cells $C(p, q) \subseteq \mathbb{R}^2$ with $p \in P_i$ and $q \in Q_i$, such that $C(p, q) := \{t \,|\, f(t) = w_i \,\|(p + t) - q\|\}$.

Theorem 4 *The decomposition of the translation space into cells $C(p, q)$ and with it the transformations minimizing $f(t)$ can be computed in $O\left(m^2 n^2\, \alpha(m^2 n^2) \log mn\right)$ time, where $\alpha(\cdot)$ is the inverse Ackermann function.*

Proof. To characterize $C(p, q)$ we fist observe, that $f(t) = w_i \,\|(p + t) - q\|$ implies $t \in \mathrm{Vor}\,(q - p, S_i(p))$, where $\mathrm{Vor}\,(a, A)$ is the Voronoi cell of the side $a \in A$ of the set A. Otherwise, another point q' of color i would be closer to p than q, implying that $f(t)$ would not by realized by p and q. There are two possible reasons why for a translation $t \in \mathrm{Vor}\,(q - p, S_i(p))$ the value of $f(t)$ might not be realized by p and q. This is either because another point $p' \in P_i$ of the same color has a larger distance to its nearest neighbor for this translation, or because the weighted distance of a closest point pair of another color $j \neq i$ exceeds $w_i\|(p+t)-q\|$, that is, $f(t) = w_i\|(p+t)-q\|$ implies

$$\forall_{1\leq j\leq k} \forall_{p'\in P_j} \min_{q'\in Q_j} w_j\|(p'+t) - q'\| \leq w_i\|(p+t) - q\|.$$

We define $B(p, q)$ as the *blocked area* for the pair p and q, i.e., all translations $t \in \mathrm{Vor}\,(q - p, S_i(p))$ for which the value of $f(t)$ is not realized by p and q due to either of the just mentioned reasons.
The characterization of the shape of $B(p, q)$ for a point

pair of the i^{th} color follows directly from the previous observation and is given as

$$B(p,q) := \bigcup_{\tilde{p}\in P_i\setminus\{p\}} \mathrm{Vor}\,(q-p, S_i(\tilde{p}) \cup S_i(p)) \;\cup$$
$$\bigcup_{\substack{1\le j\le k\\ i\ne j}} \bigcup_{\hat{p}\in P_j} \mathrm{MWVor}\,(q-p, S_i(p), S_j(\hat{p}), w_i, w_j)\,,$$

where $\mathrm{MWVor}\,(a, A, B, w, w')$ is the Voronoi cell of a point $a \in A$ in the multiplicatively weighted Voronoi diagram of the set $A \cup B$ where all points in A have weight w and all points in B have weight w'. Note, that the Voronoi cells that are united in the characterization of $B(p,q)$ correspond to the portions of the translation space for which $w_i\|(p+t)-q\|$ is not the maximal distance shortest pair. A cell $C(p,q)$ of a point pair of the i^{th} color is then fully characterized by

$$C(p,q) = \mathrm{Vor}\,(q-p, S_i(p)) \setminus B(p,q).$$

For a non empty cell that has an empty blocking area the value of $f(t)$ is minimal at the site that defined the cell and increases linear to the distance to this site. Therefore, the minimal value of $f(t)$ is either realized at a site or at the boundary of a blocking area. This means that the minima of $f(t)$ can be computed while computing the decomposition of the translation space into cells.

Let $b_{p,q}$ denote the number of Voronoi edges that contribute to $B(p,q)$.

Lemma 5 *The combinatorial complexity of $B(p,q)$ is $O\left(b_{p,q}\, 2^{\alpha(b_{p,q})}\right)$.*

Proof. By introducing polar coordinates (r,θ) with $q-p$ as the origin, the boundary of $B(p,q)$ can be seen as the upper envelope of the partially-defined, continuous, univariate functions given as the edges of the Voronoi diagrams parametrized by θ. Two Voronoi edges can intersect in at most two points. Applying the theory of Davenport-Schinzel sequences [4], this results in a complexity of $O\left(b_{p,q}\; 2^{\alpha(b_{p,q})}\right)$. □

Let $b = \sum_{1\le i\le k} \sum_{p\in P_i} \sum_{q\in Q_i} b_{p,q}$ be the total number of edges that contribute to the boundary of any blocking region.

Lemma 6 $b = O\left(m^2n^2\right)$

Proof. First fix a color i and a point $p \in P_i$. The edges e that contribute to the boundaries of any blocking area of a facet of $\mathrm{Vor}\,(S_i(p))$ result from the edges of the Voronoi diagrams

$$e \;\in\; \bigcup_{\tilde{p}\in P_i\setminus p} \mathrm{Vor}\,(S_i(p) \cup S_i(\tilde{p})) \;\cup$$
$$\bigcup_{\substack{1\le j\le k\\ i\ne j}} \bigcup_{\hat{p}\in P_j} \mathrm{MWVor}\,(S_i(p), S_j(\hat{p}), w_i, w_j)\,.$$

Let $b_{i,p}$ be the number of edges contributing to the blocking areas for the facets of $\mathrm{Vor}\,(S_i(p))$. The combinatorial complexity of a standard Voronoi diagram of n sites is $O\,(n)$ due to Euler's formula, the complexity of an multiplicatively weighted Voronoi diagram for the same number of sites however is $\Theta(n^2)$ as shown by Aurenhammer et al. [1], even if just two different weights are involved. This is the main reason, why the runtime for colored points increases compared to the monochromatic variant discussed by Huttenlocher et al. [3].
This leads to a combinatorial complexity for $b_{i,p}$ of $b_{i,p} = O\left(\sum_{1\le j\le k}\; n_j(m_i+m_j)^2\right)$. Summing over all colors i and all points $p \in P_i$ gives $b = O\left(m^2n^2\right)$. □

Lemma 5 and Lemma 6 together imply that the combinatorial complexity of the whole decomposition is $O\left(m^2n^2\, 2^{\alpha(m^2n^2)}\right)$.

The algorithm to compute the translations that minimize the directed Hausdorff distance involves two steps: First, all (multiplicatively weighted) Voronoi Diagrams have to be computed. Aurenhammer et al. [1] presented an algorithm to compute weighted Voronoi diagrams of n sites in $O\left(n^2\right)$. It takes $O\left(m^2n^2\right)$ time to compute all diagrams, as argued for Lemma 6.
In the second step, the blocking areas of all facets of all Voronoi diagrams have to be computed. As shown by Hershberger [2] the upper envelope of n partially defined functions that mutually intersect in at most s points can be computed in $O\left(\lambda_{s+1}(n)\log n\right)$ time, which leads to runtime of $O\left(m^2n^2\,\alpha(m^2n^2)\log mn\right)$, as stated in the theorem. □

References

[1] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251 – 257, 1984.

[2] J. Hershberger. Finding the upper envelope of n line segments in o(n log n) time. *Inf. Process. Lett.*, 33(4):169–174, 1989.

[3] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of voronoi surfaces and its applications. In *SCG '91: Proceedings of the seventh annual symposium on Computational geometry*, pages 194–203, New York, NY, USA, 1991. ACM.

[4] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, New York, NY, USA, 1996.

[5] H. Tan and Y.-J. Zhang. A novel weighted hausdorff distance for face localization. *Image and Vision Computing*, 24(7):656 – 662, 2006.

Computing Push Plans for Disk-Shaped Robots

Mark de Berg* Dirk H.P. Gerrits*

Abstract

Suppose we want to move a passive object along a given path, among obstacles in the plane, by pushing it with an active robot. We present two algorithms to compute a push plan for the case that the obstacles are non-intersecting line segments, and the object and robot are disks. The first algorithm assumes that the robot must maintain contact with the object at all times, and produces a shortest path. There are also situations, however, where the robot has no choice but to let go of the object occasionally. Our second algorithm handles such cases, but no longer guarantees that the produced path is the shortest possible.

1 Introduction

A fundamental problem in robotics is *path planning* [9], in which a robot has to find ways to navigate through its environment from its initial configuration to a certain destination configuration, without bumping into obstacles. Many variants of this problem have been studied, involving widely differing models for the environment, and for the robot and its movement. In *manipulation path planning* [7] the robot's goal is to make a passive object, rather than the robot itself, reach a certain destination. Several different kinds of manipulation have been studied, including grasping [7], squeezing [5], rolling [2], and even throwing [8].

The manipulation path planning problem studied here involves *pushing* [7]. In particular, we want a disk-shaped robot to push a disk-shaped object to a given destination in the plane among polygonal obstacles. Nieuwenhuisen et al. [9, 11, 12] developed a probabilistically complete algorithm for this based on the *Rapidly-exploring Random Trees* path-planning algorithm [6]. This builds a tree of reachable positions by repeatedly generating object paths and trying whether the pusher can make the object follow such a path. Thus a subroutine is needed to push the object along a given path.

Problem statement. In the plane, let P be a disk-shaped robot (of radius r_p) called the *pusher*, let O be a disk-shaped *object* (of radius $r_o > r_p$), and let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be a set of non-intersecting line segments called the *obstacles*. We're given a collision-free path τ for O consisting of k constant-complexity curves $\tau_1, \ldots, \tau_k$ called the *path sections*. We then want to compute a collision-free path σ for P such that P pushes O along τ when P moves along σ. We allow P and O to slide along obstacles, which is called a *compliant* motion. The computed path σ will be called a *push plan*. We distinguish two kinds of push plans: *contact-preserving push plans* in which the pusher maintains contact with the object at all times, and *unrestricted push plans* in which the pusher can occasionally let go of the object.

Related work. Along with the algorithm described above, Nieuwenhuisen et al. [9, 10] also developed a subroutine needed by the algorithm that solves the problem just described. They assume the object path consists of line segments and circular arcs only, and after preprocessing the n obstacles in $O(n^2 \log n)$ time into an $O(n^2)$-space data structure, they can compute a contact-preserving push plan in $O(kn \log n)$ time. It is not guaranteed that the constructed push plan is optimal in any way.

Agarwal et al. [1] considered the problem where only the final destination of the object is given, and not its path τ. For this they give an algorithm for finding a contact-preserving push plan for a point-size pusher and a unit-disk object. The algorithm discretizes the problem in two ways: the angle at which the pusher can push is constrained to $1/\varepsilon$ different values, and the combined boundary of the obstacles is sampled at m locations to give potential intermediate positions for the object. The algorithm then runs in $O((1/\varepsilon)m(m + n)\log n)$ time, but is only guaranteed to find a solution if $1/\varepsilon$ and m are large enough. The algorithm assumes the pusher can get to any position around the object at all times, which is true for their point-sized pusher but not for our disk-shaped pusher: there may be obstacles in the way.

Our results. We present a new algorithm for computing contact-preserving push plans for a given object path. It matches the $O(kn \log n)$ time needed by Nieuwenhuisen's approach, while handling path sections other than line segments and circular arcs, and without needing a $O(n^2 \log n)$-time and $O(n^2)$-space preprocessing step. We also present the first algorithms to compute *shortest* contact-preserving push plans, in $O(k^2n^2 \log(kn))$ time and $O(k^2n^2)$ space, and to to compute unrestricted push plans, in $O(kn \log(kn) + kn^2 \log n)$ time and $O(kn^2)$ space.

*Dept. of Computer Science, TU Eindhoven, the Netherlands, mdberg@win.tue.nl, dirk@dirkgerrits.com

If we assume the obstacles aren't too densely packed, these time and space bounds can be greatly improved. Both Nieuwenhuisen's and our approach for computing contact-preserving push plans then take $O((k+n)\log(k+n))$ time and $O(k+n)$ space, but the former still needs its expensive preprocessing step. For shortest contact-preserving push plans our approach takes $O((k+n)\log(k+n)+k^2\log k)$ time and $O(k+n)$ space, and for unrestricted push plans it takes $O((k+n)\log(k+n)+kn)$ time and $O(kn)$ space. For lack of space we only present the high-obstacle-density results here, and omit some proofs. Details can be found in Gerrits's MSc thesis [4].

2 Preliminaries

In the problem statement we've been deliberately vague about what pushing the object along a path entails. Before discussing our algorithms we fill in some of the details.

The push range. As mentioned before, *compliant motions* are motions where the object slides along an obstacle. Such motions are more robust in the presence of sensor inaccuracies, because the obstacle will act as a guide for the object. More importantly, this allows the pusher to achieve the same motion for the object from a continuous range of different pushing positions, called the *push range*. The pusher can then swerve around the object to avoid obstacles while still pushing the object in the desired direction. (See Figure 1(a).) With a non-compliant motion, the push range is a single pushing position depending only on the desired direction of motion for the object. If any obstacles are in the way, there simply exists no push plan for that object motion. (See Figure 1(b).)

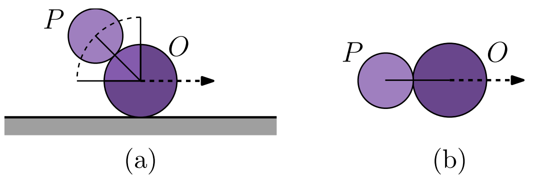


Figure 1: The push range for (a) a compliant motion, and (b) a non-compliant motion.

Friction. The exact size of the push range for compliant motions depends on the friction characteristics of the two disks and the obstacles. Nieuwenhuisen [9] describes how to compute the push range, given the friction coefficients between the disks and between the object and obstacles. Friction also affects how pushing works for non-compliant motions. Agarwal et al. [1] studied the motion of the object resulting from pushing in a straight line under simple friction assumptions.

We abstract away from compliance and friction by assuming that we can compute the push range for any position of the object along its path. We assume (as do Nieuwenhuisen and Agarwal et al.) that pushing is *quasi-static* [13], i.e. when pushing stops, the object also stops instantly. (This will never be the case in reality, but it can be closely approximated by pushing slowly, or having very high friction between the disks and the floor.)

Assumptions about the input path. We assume the given path $\tau : [0,1] \to \mathbb{R}^2$ for the object does not take it through any of the obstacles. Furthermore, we assume that τ is made up of k constant-complexity curves $\tau_1,\dots,\tau_k$. We further assume that the path sections are "well-behaved" (in a technical sense explained in Gerrits's MSc thesis [4]). The line segments and circular arcs used as path sections by Nieuwenhuisen are all well-behaved.

3 Pushing while maintaining contact

A general-purpose technique for path-planning is to translate the problem from the *work space* into the *configuration space*. The work space is the environment in which the robot has to find a path, and a *configuration* is one specific placement of the robot in this space. Each point in the configuration space corresponds to a configuration in the work space. Some configurations are invalid because the robot would intersect an obstacle and these form the *forbidden (configuration) space*. The remainder is the *free (configuration) space*, and a path through it translates back to a solution to the original path-planning problem in the work space.

To apply this technique to our problem, we first discuss what our configuration space looks like, then how to compute it and how to find a path through it.

Shape of the configuration space. In our case, a configuration is a placement of both the pusher and the object in the work space. Since the object is restricted to the path τ, and we assume that the pusher and object can maintain contact at all times (for now; we will lift this restriction in Section 4), the configuration space is two-dimensional. The point $(s,\theta) \in [0,1] \times \mathcal{S}^1$ in the configuration space will represent the configuration with the object's center at $\tau(s)$ and with θ being the *pushing angle*: the angle that the line from the pusher's center to the object's center makes with the positive x-axis. (Note that the configuration space is cylindrical, but for clarity we will depict it "flattened" as a rectangle.)

We assume that path τ does not take the object through any obstacles, so a configuration can be invalid for only two reasons: either the pusher intersects an obstacle, or the pusher is outside of the push range.

We therefore consider the forbidden space to be the union of two kinds of shapes. A *configuration-space obstacle* $\mathcal{C}_\gamma$ consists of the configurations where the pusher intersects obstacle γ. A *forbidden push range* FPR_i consists of the configurations where the object is on the interior of path section τ_i and the pusher is outside the push range. By $\mathcal{C}_{\gamma,i}$ we'll mean the restriction of $\mathcal{C}_\gamma$ to configurations with the object on path section τ_i. The forbidden space is then the union of these $k(n+1)$ shapes (n obstacles and 1 forbidden push range per path section). An example is shown in Figure 2.

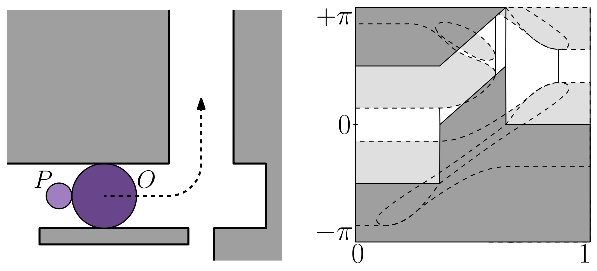


Figure 2: An example work space and its configuration space. The s-axis is horizontal, the θ-axis is vertical. Configuration-space obstacles are drawn dashed in light gray, the forbidden push range is drawn in dark gray.

Theorem 1 *The configuration space has complexity $O(kn)$, i.e. the boundary of the forbidden space consists of $O(kn)$ vertices and constant-complexity curves between them.*

Proof. Since a path section τ_i has constant complexity, so do FPR_i and $\mathcal{C}_{\gamma,i}$ for all $\gamma \in \Gamma$. We will prove that $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. It then follows that $\mathrm{FPR}_i \cup \bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$, the forbidden space for one path section, also has complexity $O(n)$, yielding $O(kn)$ in total.

The boundary of $\mathcal{C}_\gamma$ corresponds to configurations where the pusher is compliant with γ. Such pusher positions all lie at distance r_p from γ and thus form a "capsule." A point of intersection of $\mathcal{C}_{\gamma_1,i}$ and $\mathcal{C}_{\gamma_2,i}$ corresponds to a configuration where the pusher is compliant with both γ_1 and γ_2, and that pusher position must thus be the intersection of the corresponding capsules. These capsules form a collection of *pseudodisks* [3, Chapter 13], and therefore have a union complexity of $O(n)$. Thus there can only be $O(n)$ positions where the pusher would be compliant with more than one obstacle. Each of these pusher positions could show up in the configuration space more than once, since path τ_i could take the object past this point multiple times. However, this cannot happen more than $O(1)$ times, since τ_i is well-behaved. Thus $\bigcup_{\gamma\in\Gamma} \mathcal{C}_{\gamma,i}$ has complexity $O(n)$. □

Computing the configuration space. To compute the configuration space in a form that allows us to easily compute a push plan we perform the following steps:

1. Compute $\mathcal{C}_{\gamma,i}$ for all $\gamma \in \Gamma$, and all $\tau_i \in \tau$.
2. Compute FPR_i for all $\tau_i \in \tau$.
3. Take the union of these shapes to get the forbidden space.
4. Divide the free space into cells by a vertical decomposition.
5. Create the *cell graph* by directing an edge from cell c_1 to c_2 iff c_1's right boundary touches c_2's left boundary.

The running time of this approach is expressed by the following theorem:

Theorem 2 *The configuration space can be computed in $O(kn \log^2 n)$ time worst-case, or $O(kn \log n)$ expected time, both using $O(kn)$ space.*

Computing a shortest contact-preserving push plan. Not every path through the free space actually yields a push plan. If a path through the configuration space is not s-monotone then the pusher would have to pull the object on occasion. To prevent this we remove from the cell graph all cells that are not reachable from the starting configuration by a valid contact-preserving push plan. It's then fairly simple to find an arbitrary s-monotone path in linear time, resulting in a contact-preserving push plan in the time and space bounds of Theorem 2.

It is tempting to instead compute a Euclidean shortest path through the reachable cells. The cells themselves are s-monotone, so a shortest path through the remaining cells must yield a push plan. Unfortunately, a shortest path in configuration space does not necessarily minimize the pusher's movement in the work space. We can circumvent this problem by performing our computations in the work space. Each cell of the free space decomposition corresponds to a contiguous subset of the valid configurations. The pusher positions of these configurations also form a contiguous region in the work space. We could call this region the corresponding *work-space cell.* All work-space cells together form the region that P may move in to accomplish O's desired motion. Computing a shortest path [14] through this region then yields a shortest contact-preserving push plan.

Theorem 3 *Given the configuration space a shortest contact-preserving push plan can be computed in $O(k^2n^2 \log(kn))$ time and $O(k^2n^2)$ space.*

4 Pushing and releasing

Until now we've assumed the pusher can maintain contact with the object at all times. However, the situation that was depicted in Figure 2 does not admit such contact-preserving push plans. (In fact, we've proven [4] that this is the case for *any* object path with the same start and end point.) It does admit an unrestricted push plan, as can be seen in Figure 3.

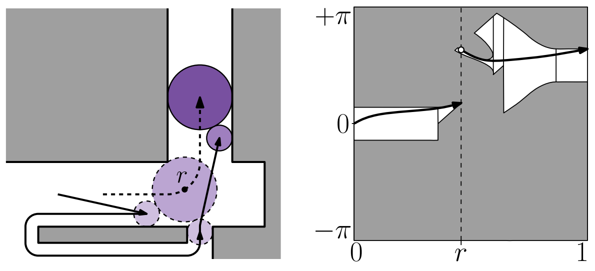


Figure 3: An unrestricted push plan for the example of Figure 2, doing one release at position r.

Canonical releasing positions. Whenever the push range is split into multiple contiguous ranges by obstacles, it may make sense for P to let go of O and try to reach one of these other positions. In the configuration space this situation corresponds to a vertical line intersecting multiple cells. In general there are infinitely many such *potential releasing positions*, thus it's infeasible to try them all. Instead we consider only vertical lines that go through a vertex of a cell or configuration-space obstacle. We call the resulting set of $O(kn)$ positions the *canonical releasing positions*. We've proved that if an unrestricted push plan exists, then there is also an unrestricted push plan where P only releases O at canonical releasing positions [4].

Computing an unrestricted push plan. Restricting ourselves to canonical releasing positions, we cannot guarantee a shortest push plan anymore. Thus we can abandon the work-space-cell approach and instead proceed as follows:

1. Compute a road map [3, Chapter 13] $\mathcal{S}$ for P among the obstacles.
2. At each canonical releasing point r, determine the set of cells intersected by the vertical line through r. Add O as an extra obstacle in $\mathcal{S}$ to get $\mathcal{S}'$, and determine for each intersected cell in which component of $\mathcal{S}'$ its pusher positions lie. Add edges in the cell graph between cells sharing a component of $\mathcal{S}'$.
3. Compute a path through this extended cell graph.
4. Convert the path into a push plan. For edges of the original cell graph this is straightforward, for the extra edges use $\mathcal{S}'$ to find a path for P.

The running time of this approach is expressed by the following theorem:

Theorem 4 *Given the configuration space an unrestricted push plan can be computed in $O(kn\log(kn)+kn^2\log n)$ time and $O(kn^2)$ space.*

References

[1] P. Agarwal, J. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *Proc. IEEE Int. Conf. Robotics & Automation*, volume 4, pages 3124–3129, 1997.

[2] H. Arai and O. Khatib. Experiments with dynamic skills. In *Proc. Japan-USA Symp. Flexible Automation*, pages 81–84, 1994.

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

[4] D. Gerrits. Designing push plans for disk-shaped robots. Master's thesis, Technische Universiteit Eindhoven, The Netherlands, 2008.

[5] K. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2–4):210–225, 1993.

[6] S. LaValle and J. Kuffner. Rapidly-exploring random trees. In B. Donald, K. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[7] M. Mason. *Mechanics of Robotic Manipulation.* Intelligent Robots & Autonomous Agents. MIT Press, 2001.

[8] M. Mason and K. Lynch. Dynamic manipulation. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, pages 152–159, 1993.

[9] D. Nieuwenhuisen. *Path Planning in Changeable Environments.* PhD thesis, Universiteit Utrecht, The Netherlands, 2007.

[10] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Path planning for pushing a disk using compliance. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots & Systems*, pages 4061–4067, 2005.

[11] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Pushing using compliance. In *Proc. IEEE Int. Conf. Robotics & Automation*, pages 2010–2016, 2006.

[12] D. Nieuwenhuisen, A. van der Stappen, and M. Overmars. Pushing a disk using compliance. *IEEE Transactions on Robotics*, 23(3):431–442, 2007.

[13] M. Peshkin and A. Sanderson. Minimization of energy in quasi-static manipulation. *IEEE Transactions on Robotics & Automation*, 5(1):53–60, 1989.

[14] M. Pocchiola and G. Vegter. Computing the visibility graph via pseudo-triangulations. In *Proc. 12th ACM Symp. Comput. Geom.*, pages 248–257, 1995.

Complexity of pleat folding

Tsuyoshi Ito[*] Masashi Kiyomi[†] Shinji Imahori[‡] Ryuhei Uehara[†]

Abstract

We introduce a new origami problem about *pleat foldings*. For a given assignment of n creases of mountains and valleys, we make a strip of paper well-creased according to the assignment at regular intervals. We use simple folding as a basic operation. More precisely, we assume that (1) paper has 0 thickness and some layers beneath a crease can be folded simultaneously, (2) each folded state is flat, and (3) the paper is rigid except at the n given creases. We also assume that each crease remembers its last folded state made at the crease. We aim to find efficient ways of folding a given mountain-valley assignment in this model. We call this problem *unit folding problem* for general patterns, and *pleat folding problem* when the mountain-valley assignment is "$MVMV\cdots$." The complexity is measured by the number of foldings and the cost of unfoldings is ignored. Trivially, we have an upper bound n and a lower bound $\log(n+1)$. We give some nontrivial upper bounds: (a) any mountain-valley assignment can be made by $\lfloor \frac{n}{2} \rfloor + \lceil \log(n+1) \rceil$ foldings, and (b) a pleat folding can be made by $O(n^\epsilon)$ foldings for any $\epsilon > 0$. We also give a nontrivial lower bound: (c) almost all mountain-valley assignments require $\Omega\left(\frac{n}{\log n}\right)$ foldings. The results (b) and (c) imply that a pleat folding is easy in the unit folding problem.

1 Introduction

Origami has recently attracted much attention as mathematics and as theoretical computer science [3]. In the computational origami, the best studied problem is the characterization of flat foldable crease patterns. When the paper and crease pattern are orthogonal, the problem is called the map folding problem, which has been well studied by Arkin et al. [2]. Roughly speaking, (1) the problem in 1D can be solved in linear time; it can be determined in $O(n)$ time whether or not a given strip of paper with n creases is flat foldable, and (2) the problem in 2D is NP-complete; to determine whether or not a given zig-zag form paper with n creases is flat foldable is NP-complete (see [2] for the details).

We deal with a simpler problem. The input of our problem is a strip of paper of length $n+1$ with n creases placed at regular intervals on the paper. That is, we assume that the paper corresponds to an interval $[0..n+1]$, n creases are placed at each integer point i with $1 \leq i \leq n$, and each integer point is assigned to M (mountain) or V (valley) which is the target folded state. We call it MV *assignment* for short. We aim to fold the paper and make it "well-creased" according to the assignment. Precisely, each integer point *memorizes* its last folded state (M or V) made at the point, and we have to make the creases' folded states fit the given assignment.

A typical example is *pleat folding* that is one of basic tools for origami design (Figure 1; the angel is made from just one square paper without any cutting). The MV assignment for the pleat folding is in the form "$MVMV\cdots$." From the industrial point of view, pleat folding has many applications including clothes, curtains, etc.

We employ *simple folding* defined in [2] (see also [3, Section 14.1]). More precisely, our *origami* is as follows; the paper has 0 thickness, some layers beneath a crease can be folded simultaneously, each folded state is flat, and the paper is rigid except at given n creases. Clearly, this folding problem has a trivial solution (or upper bound of the number of foldings); we can fold any MV assignment by just folding n creases independently. On the other hand, we have a trivial lower bound of the number of foldings; we have to fold at least $\lceil \log_2(n+1) \rceil$ times to make n creases (by folding repeatedly at the center).

For a given string s of length n in $\{M,V\}^n$, the *unit folding problem* is to obtain the well-creased paper following s. We call it *pleat folding problem* if the MV assignment is "$MVMV\cdots$." The *complexity* is measured by the number of foldings, and the number of unfoldings is ignored. We first show the following nontrivial upper bound of the number of foldings to solve any unit folding problem:

Theorem 1 *The unit folding problem of n creases can be solved by $\lfloor \frac{1}{2} n \rfloor + \lceil \log(n+1) \rceil$ foldings for any MV assignment*[1].

[*]School of Computer Science, McGill University, tsuyoshi@cs.mcgill.ca

[†]School of Information Science, JAIST, {mkiyomi,uehara}@jaist.ac.jp

[‡]Graduate School of Information Science and Technology, The University of Tokyo, imahori@mist.i.u-tokyo.ac.jp

[1]In this paper, we assume the base of logarithm is two unless specified otherwise.

Figure 1: An "angel" designed and folded by Takashi Hojyo on the cover of [1]. (`http://www.origami.gr.jp/Magazine/Index/103-108.html`)

One may think that the pleat folding problem is the most difficult in the unit folding problems. However, this is not the case. We show an efficient way to obtain the pleat foldings.

Theorem 2 *For any positive real number $\epsilon > 0$, the pleat folding problem of n creases can be solved by $O(n^\epsilon)$ foldings for sufficiently large n.*

On the other hand, we can show a lower bound of the general unit folding problem.

Theorem 3 *For the unit folding problem, almost all MV assignments with n creases require $\Omega\left(\frac{n}{\log n}\right)$ foldings.*

Theorems 2 and 3 imply that the pleat folding problem is not the most difficult unit folding problem.

In origami, "unfolding" is much easier than "folding." Hence we ignore the cost of unfoldings. However, even if we count the number of unfoldings in addition to foldings, all the results hold with small changes within constant factors.

2 Models of foldings and unfoldings

The input to a unit folding problem consists of a string of length n over an alphabet $\{M, V\}$. The ith letter (M or V) corresponds to the *ith assignment* at the ith crease which is placed at the integer point i in the interval $[0..n+1]$. In general, we have several ways when we fold the paper at the point i. The following simple fold models are proposed by Arkin et al. [2] (see also [3, p. 225]):

One-layer simple fold model: We always fold the top layer of paper.

All-layers simple fold model: We simultaneously fold all layers of paper under the crease point.

Some-layers simple fold model: We fold some layers beneath the crease point.

We measure the complexity of a folding algorithm by the number of simple foldings made by it, and ignore the cost of unfoldings. In the one-layer simple fold model, we always fold exactly one crease. Thus, we cannot improve the trivial upper bound n for the unit folding problem. Therefore, hereafter, we only consider the all-layers simple folding and some-layers simple folding. We additionally introduce the following three unfolding models, which define the allowable unfolding operations.

All-unfold model: Once we decide to unfold, the paper is unfolded completely.

Reverse-unfold model: We can rewind any number of the last foldings as far as we can. This is useful to consider some upper and lower bounds.

General-unfold model: For a folded state s, we can obtain another folded state t by the one general unfolding operation if s can be obtained from t by consecutive some-layers simple foldings.

Of course, the general-unfold model is the most natural and the strongest model. This paper mainly studies the first two unfold models, which are natural restrictions of the general-unfold model.

A folding algorithm will be called *end-free* if it can be applied on the paper with n creases even if we extend both endpoints of the paper to infinity.

3 Upper bounds

We first prove Theorem 1, which is the basic tool of this paper.

Proof. (of Theorem 1) To simplify the proof, we first suppose that $n = 2^k - 1$. The strategy is simple; (1) fold the paper in half k times, (2) unfold all, and (3) fix each crease if it is not folded correctly.

On step (1), the folding direction is determined by the majority at the point. That is, we see the point i, which is the center of the current paper, check the all directions of the desired creases on the point i, and take majority. (We note that half creases are turned over from the original direction, but fixing this is easy; the creases are alternately turned over and that can be checked from 1 to n.) Hence step (1) makes at least half of creases to be folded correctly. Thus in step (3), the number of foldings is bounded above by $\lfloor \frac{n}{2} \rfloor$ (the rounding comes from the first folding that always succeeds). Therefore we have the theorem.

When $n \neq 2^k - 1$, we just add an imaginary paper to make the length of the paper $n' = 2^k - 1$ with $\min_k n < 2^k - 1$, and apply the algorithm on this paper of length n'. □

We note that (a) the algorithm in the proof is end-free, (b) it works in the all-layers simple fold model and hence some-layers simple fold model, and (c) it works in all the unfold models since all unfoldings in steps (2) and (3) can be done by all-unfoldings.

We call a unit folding problem *mountain folding problem* if the MV assignment is "$MMM \cdots$." We next show a simple proposition which gives us a strong intuition to consider the pleat folding.

Proposition 4 *Let A' be any algorithm for the mountain folding problem in any model, and $f_0(n)$ be the number of foldings by A' to make n mountain foldings. Then we can solve the pleat folding problem in the model with $f_0(\lfloor n/2 \rfloor) + f_0(\lceil n/2 \rceil)$ foldings.*

Proof. For the MV assignment "$MVMV \cdots$," we first fold all $f_0(\lceil n/2 \rceil)$ mountains by A'. Next we reverse the paper and fold all $f_0(\lfloor n/2 \rfloor)$ valleys by A' again. □

Now we turn to the upper bounds of the number of foldings for solving the pleat folding problem. First, we focus on the all-layers simple fold and reverse-unfold (or all-unfold) model. In this weak model, we show a weaker upper bound. By Proposition 4, we can focus on the mountain folding problem instead of the pleat folding problem. This will help us to have an intuition.

Lemma 5 *The mountain folding problem of n creases can be solved by $O(n^{\log \phi}) \sim O(n^{0.69})$ foldings in the all-layers simple fold and all-unfold model, where $\phi = \frac{1+\sqrt{5}}{2}$.*

Proof. To simplify the proof, we suppose that $n = 2^k$. The basic strategy is similar to the algorithm in the proof of Theorem 1. In step (0), we fold the paper at point 1 and make the length of the paper 2^k. The next two steps (1) and (2) are the same; we fold the paper in half k times and unfold completely. Now we focus on the MV pattern made by the two steps (1) and (2). First, we fold the paper in half, and obtain the correct crease at the center point. Second, we fold the paper in half, and obtain one correct crease, and one incorrect crease. Third, we fold the paper in

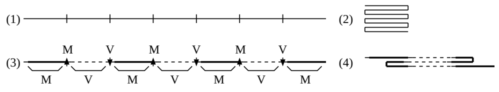


Figure 2: Mountain folding by zig-zag method

half, and obtain two correct creases, and two incorrect creases. In ith folding with $i > 1$, we obtain 2^{i-2} correct creases and 2^{i-2} incorrect creases that appear on the paper alternately. The 2^{i-2} incorrect creases are placed at regular intervals. Thus, we can use our end-free algorithm recursively on these 2^{i-2} creases in step (3). Let $f_1(n)$ be the number of the foldings of this recursive algorithm. It is easy to see that $f_1(1) = 1$, $f_1(2) = 2$, $f_1(3) = 3$, and $f_1(4) = 4$. By the above observation, we have

$$f_1(2^k) = 1+k+f_1(2^{k-2})+f_1(2^{k-3})+\cdots+f_1(2)+f_1(1).$$

By the fact that $f_1(2^k) - f_1(2^{k-1})$ equals $f_1(2^{k-2})+1$, we have the following relationship.

$$(f_1(2^k) + 1) = (f_1(2^{k-1}) + 1) + (f_1(2^{k-2}) + 1).$$

Thus $(f_1(2^k) + 1)$ gives the Fibonacci numbers for k. (We define the Fibonacci numbers F_i by $F_0 = 0$, $F_1 = 1$ and $F_i = F_{i-1}+F_{i-2}$ for $i > 1$. It is well known that $F_i = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} = O(\phi^n)$, where $\phi = \frac{1+\sqrt{5}}{2}$.) Using initial conditions $f_1(2^0) = 1$, $f_1(2^1) = 2$, and $f_1(2^2) = 4$, we have $f_1(2^k)+1 = F_{k+3}$. Hence we have $f_1(n) = f_1(2^k) = O(\phi^k) = O\left(\phi^{\log n}\right) = O\left(n^{\log \phi}\right)$, which completes the proof. □

Next, we show that the number of foldings can be reduced by using the stronger model.

Lemma 6 *For any fixed $\epsilon > 0$, the mountain folding problem of n creases can be solved by $O(n^\epsilon)$ foldings for sufficiently large n in the some-layers simple fold and reverse-unfold models.*

Proof. In the algorithm of the proof of Lemma 5, "folding in half" is the basic operation. We change this to "folding in zig-zag form of p segments" (Figure 2). Roughly speaking, the algorithm first folds the paper into zig-zag form of p segments of the same length by making $p - 1$ alternating foldings at regular intervals (in Figure 2(1) and (2) for $p = 7$). It recursively calls itself with the folded paper of length n/p. After the recursive call, it unfolds at the folded $p - 1$ crease points. Then around $p/2$ segments have been mountain folded, and $p/2$ segments have been valley folded (in Figure 2(3), thick lines and dashed lines indicate the segments of the mountain and valley foldings, respectively). The algorithm then piles up the valley segments exactly by folding at the center points of the mountain segments (see Figure 2(4)).

The algorithm again calls itself recursively with the local area of length n/p that consists of all and only the valley segments. By the recursive call, half of the valley segments are corrected, and still half of them are not corrected. (The creases of the center points of the mountain segments are fixed here.) Hence the algorithm again piles up the valley segments and recursively calls itself again. The algorithm repeats this process and finally obtain the desired mountain folded state.

The algorithm runs in the some-layers simple fold and reverse-unfold model, and the correctness of the algorithm is easy. Hence we analyze its complexity. To simplify the proof, we suppose that $n = p^k - 1$ and $p = 2^q - 1$. (We let q be an arbitrary positive integer, and hence p is odd and n is even.) Let $f_2(n)$ be the number of foldings made by the algorithm. By a careful analysis, we have $f_2(0) = 0$, $f_2(x) \leq x$ and

$$f_2(n) = q{\cdot}f\left(\frac{n+1}{p} - 1\right) + p - 2(q-2) + \frac{3(p+1)}{2}\sum_{\ell=2}^{q-1}\frac{1}{2^{\ell-1}}.$$

By a simple calculation, we obtain

$$\begin{aligned} f_2(n) &< q \cdot f\left(\frac{n+1}{p} - 1\right) + \frac{5p+11}{2} \\ &< (5p+11)q^{k-2} + p - 1 = O(p \cdot q^{k-2}). \end{aligned}$$

Since $q = \log_2(p+1)$ and $k = \log_p(n+1)$, we have $p \cdot q^{k-2} \sim p \cdot n^{\frac{\log\log p}{\log p}}$. Hence, letting p be a sufficiently large constant, we have $\frac{\log\log p}{\log p} < \epsilon$ and $p \cdot q^{k-2} = O(n^\epsilon)$, which completes the proof. $\square$

By Proposition 4 and Lemma 6, we have Theorem 2.

4 Lower bounds

We have to clarify the unfold model to state a lower bound. In fact, Theorem 3 holds for both of the all-unfold model and the reverse-unfold model, but we have no results about general-unfold model.

Lemma 7 *All but $o(2^n)$ MV assignments of n creases require $\Omega\left(\frac{n}{\log n}\right)$ foldings in all of the three folding models and both of the all-unfold model and the reverse-unfold model.*

Proof. The lemma is obtained by a simple counting argument. Since the some-layers simple fold model is at least as powerful as the one-layer and the all-layers simple fold models, we only consider the some-layers simple fold model. Similarly, we only discuss the reverse-unfold model. Suppose that we make at most k foldings. At each folding, there are two choices for the direction of folding (mountain or valley). There are at most n choices for the set of positions of the folding by considering the bottom-most layer of a valley folding or the top-most layer of a mountain folding. There are at most $n+1$ choices for how many unfolding steps we rewind just after the last folding. In total, the number of the possible MV assignments made by at most k foldings is bounded by $\sum_{i=0}^{k}(2n(n+1))^i \leq (2n(n+1)+1)^k < (2(n+1)^2)^k$. If we let $k = \lfloor\frac{n}{4(1+\log_2(n+1))}\rfloor$, then $(2(n+1)^2)^k < 2^{n/2} = o(2^n)$. Therefore all but $o(2^n)$ MV assignments require at least $\lfloor\frac{n}{4(1+\log_2(n+1))}\rfloor = \Omega\left(\frac{n}{\log n}\right)$ foldings in the reverse-unfold model (hence also in the all-unfold model). $\square$

Lemma 7 implies Theorem 3. Lemma 7 also implies that if we choose an MV assignment of n creases uniformly at random, the expected number of foldings required to make it is $\Omega\left(\frac{n}{\log n}\right)$ in the all-unfold model and the reverse-unfold model.

5 Conclusion

There are several open problems.

By Theorem 3, a random MV assignment requires $\Omega\left(\frac{n}{\log n}\right)$ foldings with high probability. Finding a specific assignment which requires $\Omega\left(\frac{n}{\log n}\right)$ foldings will give some new insight on which assignments are easy to fold and which are not. Is there any MV assignment that requires $\Omega(n)$ foldings?

Lemma 7 deals with general MV assignments. When we focus on the pleat folding problem, is there a nontrivial lower bound for this specific problem? Especially, can we make pleat foldings in poly-log foldings?

From the practical point of view, developing better algorithm for the pleat folding problem is interesting, especially, in general-unfold model. Up to now, we have no idea for dealing with general-unfold model.

Acknowledgments

The authors thank to Takashi Hojyo who gives a permission to use his "angel" for our presentation of this research. The authors are grateful to Erik Demaine, Joseph O'Rourke, and Ron Graham for their helpful comment and suggestion at CCCG 2008.

References

[1] *Origami Tanteidan*, Number 107. Japan Origami Academic Society, 2008.

[2] E. Arkin, M. Bender, E. Demaine, M. Demaine, J. Mitchell, S. Sethia, and S. Skiena. When Can You Fold a Map? *Computational Geometry: Theory and Applications*, 29(1):23–46, 2004.

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

Modelling rigid origami with Quaternions

Weina Wu [*] Zhong You [*]

Abstract

This paper presents a new tool, namely the quaternion rotation sequence (QRS) method, for modeling rigid origami. An origami pattern can be described by a rotation vector model consisting of a group of characteristic vectors. A sequential rotation of these vectors leads to a system of the closure equations represented by quaternions. The solutions of the equations provide configuration information of a pattern. The method has been successfully applied to several types of single vertex and multi-vertex patterns. We have shown that it can be used to effectively track the entire rigid folding process of initially flat or non-flat patterns, and thereby provide judgment of rigid-foldability and flat-foldability of the pattern.

1 Introduction

Rigid origami is a restrictive type of origami where all panels of a crease pattern are not allowed to stretch or bend during folding. The creases are therefore comparable to rotational hinges of a mechanism. The concept of rigid origami has great potentials in engineering. A well-known example is the Mirua-ori, which has been used for deployable solar panels and for manufacturing folded cores for sandwiched composites. It is known that essentially everything can be folded by a continuous folding procedure [1], yet the folding is not necessarily rigid. Only those geometrically compatible patterns can be folded rigidly.

A number of mathematical methods have been developed to study rigid origami, including a method based on spherical trigonometry and Kawasaki's Theorem, which examines whether a flat-foldable pattern can be folded rigidly [2]; the method employing Gaussian curvature theory originally proposed by Huffman [3], which judges a pattern as rigid if Gaussian curvature at any point of the pattern is zero; and the matrix model method proposed by Belcastro and Hull [4], where a certain configuration of a pattern is modelled as a particular successive transformation procedure of creases and panels, and a loop closure equation system is thereby established as the necessary condition for rigidity of the configuration. In spite of validity of those methods, they still bear some limitations in application. For instance, the first method is not applicable to non-flat-foldable patterns. The second is suitable for rigidity judgment at a particular configuration but inconvenient for configuration calculations. The third one can deal with more situations, yet is hardly employed on those non-flat-foldable 3D patterns. In fact, the third method bears similarity to the matrix method in loop mechanism analysis using the Denavit and Hartenberg notation [5].

In this paper, a rotation vector model consisting of a group of characteristic vectors is introduced. Then the quaternion rotation sequence (QRS) method, which represents the model via quaternions, is proposed and applied to three different types of patterns to verify its effectivity. It shows that the QRS method can effectively track the entire rigid folding procedure of an initially flat or non-flat pattern, as well as provide judgment of its rigid foldability and flat foldability. Moreover, the QRS method is applicable to not only the single-vertex pattern but also the multi-vertex pattern.

2 The rotation vector model

Consider a typical single-vertex pattern SV1, which is flat before folding as in Figure 1(a). In this paper, use solid lines for mountain creases and dashed lines for valley creases in the plot of pattern layout. Its creases, or called edges, are enumerated from E_1 to E_4 anticlockwise, and panels from P_1 to P_4. Denote by $\alpha_{i,j}$ the layout angle subtended by edge E_i and E_j (i=1, 2, 3, 4, j=2, 3, 4, 1). For pattern SV1, it has $\alpha_{1,2} = 90^\circ$, $\alpha_{2,3} = 135^\circ$, $\alpha_{3,4} = 90^\circ$ and $\alpha_{4,1} = 45^\circ$. Figure 1(b) shows an arbitrary 3D configuration of the pattern after certain extent of rigid folding, where its characteristic vectors are established. They include edge vectors $\vec{e}_i$ along edges E_i (i=1, 2, 3, 4) and normal vectors $\vec{n}_i$ perpendicular to panels P_i (i=1, 2, 3, 4). As shown in Figure 2, these vectors are related in a rotation sequence, which can be decomposed into eight sequential steps, where alternately, an $\vec{n}_i$ or $\vec{e}_i$ derived from the previous step acts as the rotation axis, around which $\vec{e}_i$ rotates to $\vec{e}_{i+1}$, or $\vec{n}_{i-1}$ to $\vec{n}_i$. The corresponding rotation angles are alternately $\alpha_{i,j}$ or γ_i. Denote by γ_i the folding angle between normal vectors of adjacent panels which intersect at edge E_i, and it follows the right-handed screw rule. Note that the last two steps lead to the initial vectors $\vec{e}_1$ and $\vec{n}_1$, respectively, indicating that these vectors form a closed-loop rotation sequence. To get a more vis-

[*]Department of Engineering Science, Oxford University, UK, `weina.wu@eng.ox.ac.uk, zhong.you@eng.ox.ac.uk`

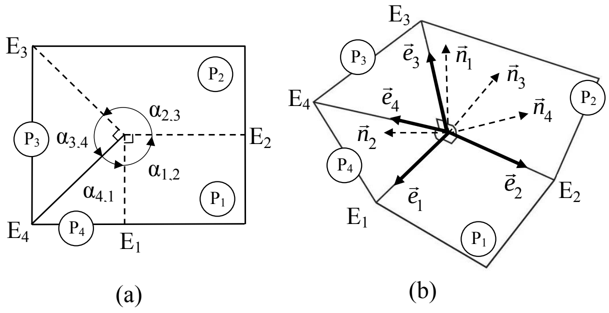


Figure 1: Crease pattern SV1, (a) the initial flat state, (b) its characteristic vectors at an assumed configuration after rigid folding

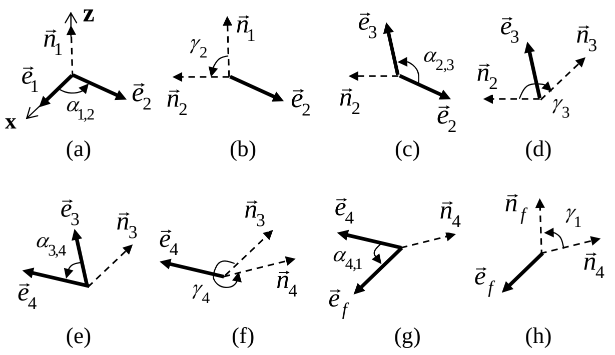


Figure 2: The closed-loop rotation sequence of the characteristic vectors decomposed into eight steps

ible mathematical description of the 3D configuration, we use dihedral angles $\theta_i = \pi - \gamma_i$ to replace γ_i. For convenience, let us stipulate that dihedral angles are independent of the rotational direction, and all dihedral angles of a pattern are measured from the same side throughout folding. Actually by only observing from one side of the pattern, all creases fall into two categories, i.e. mountain and valley. Therefore if assuming a pattern is initially flat and finally folded flat, the maximum attainable range of θ_i around each type of creases is $0 \leq \theta_{Valley} \leq 180°$ and $180° \leq \theta_{Mountain} \leq 360°$.

The rotation vector model associates the pattern information, i.e. the layout angles, with the configuration information, i.e. the folding or the dihedral angles. It is essentially identical to a closed-loop spherical linkage model [6]; thus its degree of freedom (DOF) can be determined on the same rule as that for the linkage.

$$\text{DOF} = 3(m - j - 1) + \sum_{i=1}^{j} f_i \tag{1}$$

where m is the number of edge vectors, j the number of normal vectors, and f_i the DOF of edge $\text{E}_i(i=1, 2, \dots j)$ as a rotation axis. For example, for the pattern SV1, with $m = j = 4$ and $f_i = 1$ (i=1, 2, 3, 4), its freedom is 1.

3 The quaternion rotation sequence (QRS) method

Quaternions were originally proposed by William Rowan Hamilton in 1843. The basic quaternion algebra can be found in [7] and [8]. Compared with ordinary algebra, the most special part is that quaternion multiplication doesn't obey the law of commutativity, which actually facilitates its ability to represent arbitrary 3D rotations. The proposed quaternion rotation sequence (QRS) method employs quaternions to represent the rotation vector model, and solves for the unknown configuration angles through optimization.

Consider pattern SV1 at a configuration shown in Figure 1(b). In a 3D Cartesian coordinate frame shown in Figure 2(a), we can write $\vec{e}_1 = \{1, 0, 0\}$ and $\vec{n}_1 = \{0, 0, 1\}$, which are the initial vectors to start the rotation sequence. The rotation around $\vec{n}_1$ in Figure 2(a) can be represented by a quaternion rotation operator

$$\tilde{q}^{\vec{n}_1} = \{\cos(\alpha_{1,2}/2), \quad \sin(\alpha_{1,2}/2)\vec{n}_1\} \tag{2}$$

The resultant vector $\vec{e}_2$ is the vector part of the quaternion $\tilde{e}_2$,

$$\tilde{e}_2 = \tilde{q}^{\vec{n}_1}\, \tilde{e}_1\, (\tilde{q}^{\vec{n}_1})' \tag{3}$$

When $\vec{e}_2$ is known, the rotation around it in Figure 2(b) can be expressed by

$$\tilde{q}^{\vec{e}_2} = \{\cos((\pi - \theta)/2), \quad \sin(\pi - \theta/2)\vec{e}_2\} \tag{4}$$

The resultant vector $\vec{n}_2$ is the vector part of the quaternion $\tilde{n}_2$,

$$\tilde{n}_2 = \tilde{q}^{\vec{e}_2}\, \tilde{n}_1\, (\tilde{q}^{\vec{e}_2})' \tag{5}$$

The following rotations and resultant vectors at subsequent steps in Figure 2 can be represented likewise, until two initial vectors $\vec{e}_1$ and $\vec{n}_1$ are reconstructed, which are denoted by $\vec{e}_\text{f}$ and $\vec{n}_\text{f}$, respectively. The following loop closure equation system can therefore be obtained.

$$\begin{cases} \vec{n}_\text{f} &= \vec{n}_1 \\ \vec{e}_\text{f} &= \vec{e}_1 \end{cases} \tag{6}$$

where a total of *six* simultaneous equations are included, all expressed in terms of $\alpha_{i,j}$ and θ_i. Normally $\alpha_{i,j}$ are known for a given pattern. Considering the DOF of the system according to Equation 1, the same number of dihedral angles is required as the input; whereas the remaining unknown dihedral angles can be solved.

Here a constrained nonlinear optimization algorithm, the sequential quadratic programming (SQP)

]

method, is employed as the solver, where the optimization goal is to minimize the difference between two sides of each equation, and the range constraint imposed on each variable comes from whether it is around a mountain or a valley crease. Despite severe nonlinearity of the equation, it has shown that for most patterns the proper initial estimates of its unknowns can be easily picked up from a preliminary optimization which just uses arbitrary initial estimates. Actually piecewise initial estimates can be defined for different input ranges, which further eases giving the initial estimations. Given the input a sequence of values, changing configurations during a folding procedure can be described. The verification rule for those optimization results is that the successive values of each output angle must not be fluctuating. Fluctuating angle values physically correspond to a discontinuous folding procedure, thus incorrect.

Case 1: for an initially flat (2D) pattern
For pattern SV1 with a single DOF, assume the input as θ_1 decreasing monotonically from 180° to 0 at the step of 1° during the entire folding. The final optimization results of the other three dihedral angles θ_2, θ_3 and θ_4 are are shown in Figure 3(a). These smooth angle values describe all configurations reached by the pattern throughout folding in the given input range, which indicate that such rigid folding procedure is continuous and thus physically meaningful. Therefore it can be concluded that the pattern is rigid foldable for $0 \leq \theta_1 \leq 180°$ and also flat foldable.

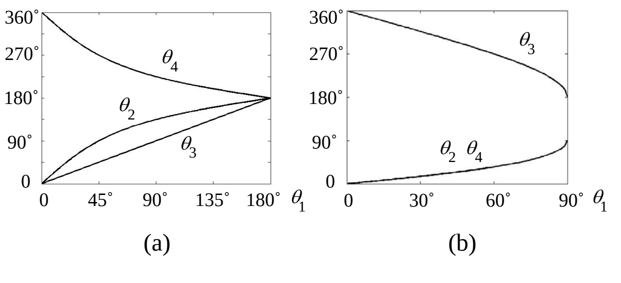


Figure 3: Output angles for (a) pattern SV1 and (b) pattern SV2 throughout folding via the QRS method

Case 2: for an initially non-flat (3D) pattern
The 3D pattern SV2 in Figure 4(a) has $\alpha_{1,2} = 90°$, $\alpha_{2,3} = 45°$, $\alpha_{3,4} = 45°$ and $\alpha_{4,1} = 90°$. Its dihedral angles measured from the inward side of the pattern are $\theta_1 = \theta_2 = \theta_4 = 90°$ and $\theta_3 = 180°$. If taking the given configuration as the initial and assuming SV2 is finally folded flat, to track its folding procedure, the required input for the this single DOF system can be selected as θ_1 $(0 \leq \theta_1 \leq 90°)$ changing at the step of 1°. Correspondingly, the maximum attainable ranges of the other three dihedral angles are $0 \leq \theta_2 \leq 90°$,

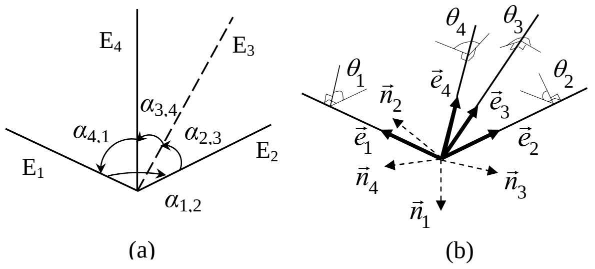


Figure 4: Pattern SV2, (a) the initial 3D state, (b) its characteristic vectors at an assumed configuration after rigid folding

$180° \leq \theta_3 \leq 360°$ and $0 \leq \theta_4 \leq 90°$. Assume an arbitrary configuration of the pattern during rigid folding as in Figure 4(b) to establish the loop closure equation system. The final verified optimization results of the output angles θ_1, θ_2 and θ_3 are shown in Figure 3(b). They again confirm the rigid foldability of the pattern in the given input range by showing that rigid folding of the pattern is a continuous process. θ_1 with values greater than 90° has also been tried, yet the derived results are found all fluctuating and thus physically unrealistic. Therefore SV2 cannot be folded rigidly when $\theta_1 > 90°$.

Case 3: for a multi-vertex pattern
The QRS method is applicable to the multi-vertex pattern via a decomposition strategy. At first select a loop of creases connecting multiple vertices, around each of which the multi-vertex system is decomposed into multiple single-vertex subsystems. Then these subsystems are analyzed sequentially along the loop in one direction, each as an independent single-vertex system with the input transferred from its previous subsystem, i.e. as their common dihedral angle. After all subsystems along the loop are analyzed, check whether the dihedral angle common in the first and the last subsystems has the same values. If so, global compatibility of the entire system can be confirmed. With this strategy, the real complexity of the multi-vertex system is actually averted. For example, if each subsystem has the single DOF, the whole system needs only one input as well.

Consider two multi-vertex crease patterns MV1 and MV2 in Figure 5, where all edges and vertices are enumerated and a crease loop through four vertices is encircled for each pattern. For simplicity, here take the crease layout around all vertices in MV1 or MV2 as the same. One set of layout angles for each pattern are marked in Figure 5. Assume that the input θ_1 for both patterns changes in the range of $0 \leq \theta_1 \leq 180°$ at the step of 1°. Along the crease loop anticlockwise, all single-vertex subsystems at V$_1$, V$_2$, V$_3$ and V$_4$ are analyzed sequentially, and the results from all subsys-

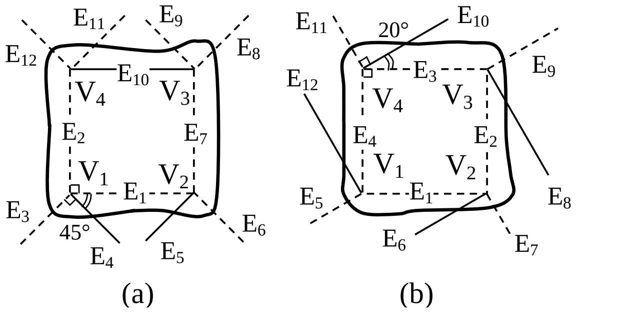


Figure 5: Multi-vertex patterns (a) MV1 and (b) MV2. A loop enclosing multiple vertices is drawn on each pattern in the bold line.

tems are summarized. Some typical results from pattern MV1 are displayed in Table 1, and from pattern MV2 in Table 2. As shown in Table 1, throughout folding the values of θ_2 are the same in the first and the last subsystems around vertex V_1 and V_4, respectively; while in Table 2, the values of θ_4 in the first and the last subsystems are quite different. Since the the crease loop must be closed, it can be concluded that pattern MV1 is globally compatible, able to be folded rigidly as a whole for $0 \leq \theta_1 \leq 180°$ and be completely folded flat given the right crease lengths; while pattern MV2 cannot be folded rigidly at all.

Table 1: Dihedral angles of pattern MV1 at some configurations during folding (°)

V_1	θ_2	**23.9**	**46.1**	**141.7**	**171.6**	**175.8**
	θ_3	10.0	20.0	100.0	160.0	170.0
	θ_4	336.1	313.9	218.3	188.4	184.2
	θ_1	*10.0*	*20.0*	*100.0*	*160.0*	*170.0*
V_2	θ_1	*10.0*	*20.0*	*100.0*	*160.0*	*170.0*
	θ_5	336.1	313.9	218.3	188.4	184.2
	θ_6	10.0	20.0	100.0	160.0	170.0
	θ_7	23.9	46.1	141.7	171.6	175.8
V_3	θ_7	23.9	46.1	141.7	171.6	175.8
	θ_8	54.0	91.6	163.6	176.5	178.3
	θ_9	23.9	46.1	141.7	171.6	175.8
	θ_{10}	306.0	268.4	196.4	183.5	181.7
V_4	θ_{10}	306.0	268.4	196.4	183.5	181.7
	θ_{11}	23.9	46.1	141.7	171.6	175.8
	θ_{12}	54.0	91.6	163.6	176.5	178.3
	θ_2	**23.9**	**46.1**	**141.7**	**171.6**	**175.8**

4 Summary

So far we have presented a QRS method for modeling rigid origami. The pattern and configuration information of an origami pattern are described using a group of characteristic vectors. Through a closed-loop rotation sequence of the identified characteristic vectors, represented by quaternions, we are able to establish a system of closure equations. Full angular descriptions of attainable configurations of a pattern

Table 2: Dihedral angles of pattern MV2 at some configurations during folding (°)

V_1	θ_4	**7.0**	**44.0**	**101.0**	**151.7**	**172.9**
	θ_{12}	350.0	300.0	240.0	200.0	185.0
	θ_5	7.0	44.0	101.0	151.7	172.9
	θ_1	*10.0*	*60.0*	*120.0*	*160.0*	*175.0*
V_2	θ_1	*10.0*	*60.0*	*120.0*	*160.0*	*175.0*
	θ_6	345.8	281.0	224.0	194.1	183.5
	θ_7	10.0	60.0	120.0	160.0	175.0
	θ_2	14.2	79.0	136.0	165.9	176.5
V_3	θ_2	14.2	79.0	136.0	165.9	176.5
	θ_8	339.8	260.7	211.6	189.9	182.5
	θ_9	14.2	79.0	136.0	165.9	176.5
	θ_3	20.2	99.3	148.4	170.1	177.5
V_4	θ_3	20.2	99.3	148.4	170.1	177.5
	θ_{10}	331.4	241.5	202.4	186.9	181.7
	θ_{11}	20.2	99.3	148.4	170.1	177.5
	θ_4	**28.6**	**118.5**	**157.6**	**173.1**	**178.3**

during folding can be derived by solving these equations. Using single vertex and multi-vertex origami patterns, we demonstrate that the QRS method can effectively track the entire rigid folding process of initially flat or non-flat patterns, and thereby provide judgment for rigid-foldability and flat-foldability. As to the multi-vertex patterns, we also attempted using dual quaternions and found that it is also effective. However, this is not to be presented here because of the page limit. The presented method can be used in engineering applications such as designing space panel structures and some packaging where rigid folding is required or preferable.

References

[1] Demaine, E.D., Devadoss, S. L., Mitchell, J. S. B., and O'Rourke, J. *Continuous foldability of polygonal paper.* Proceeding of 16th Canadian Conference on Computational Geometry, Montreal, Quebec, 2004: 64-67.

[2] Hull, T. *Project origami.* A K Peters, Ltd, Wellesley, Massachusetts, 2006: Activity 22, 229-235.

[3] Huffman, D. *Curvature and crease: a primer on paper.* IEEE Transactions on Computers, C-25, No. 10, 1976: 1010-1019.

[4] Belcastro, S., and Hull, T. C. *Modeling the folding of paper into three dimensions using affine transformations.* Linear Algebra and its Applciations, Vol. 348, 2002: 273-282.

[5] Beggs, J S. Advanced Mechanism, Macmillan Company, New York, 1966.

[6] Waldron, K. J., and Kinzel, G. L. *Kinematics, dynamics, and design of machinery.* New York, Chichester: Wiley, 1999: Chapter 2.

[7] Kuipers, J. B. *Quaternions and Rotation Sequences: A Primer With Applications to Orbits, Aerospace, and Virtual Reality.* Princetion University Press, USA, 2002.

[8] http://www.euclideanspace.com/maths/algebra/.

Source Unfoldings of Convex Polyhedra with respect to Certain Closed Polygonal Curves

Jin-ichi Itoh[*] Joseph O'Rourke[†] Costin Vîlcu[‡]

Abstract

We extend the notion of a source unfolding of a convex polyhedron $\mathcal{P}$ to be based on a closed polygonal curve Q in a particular class rather than based on a point. The class requires that Q "lives on a cone" and therefore has a simple development in the plane. The class includes simple closed convex curves and curves that are the intersection of $\mathcal{P}$ with a plane. Cutting a particular subset of the cut locus of Q (in $\mathcal{P}$) leads to a non-overlapping unfolding of the polyhedron. This gives a new general method to unfold the surface of any convex polyhedron to a simple, planar polygon.

1 Introduction

Two general methods were known to unfold the surface $\mathcal{P}$ of any convex polyhedron to a simple polygon in the plane: the source unfolding and the star unfolding [DO07], both with respect to a point $x \in \mathcal{P}$. In [IOV08] we defined a third general method, the star unfolding with respect to a simple closed "quasi-geodesic loop" Q on $\mathcal{P}$. Here we extend the source unfolding to a class of closed polygonal curves Q that includes quasigeodesic loops but is considerably wider.

Q divides $\mathcal{P}$ into two closed "halves," P_1 and P_2. (We sometimes use P to represent either half.) We establish that the source unfolding w.r.t. Q for each half P_i unfolds without overlap, and that cutting a subset of the cut locus of Q in one half, and some additional cuts in the other half, leads to a single piece. Because Q in the class exist (in abundance) for any $\mathcal{P}$, this leads to another general method to unfold any convex polyhedron to a simple, planar polygon.

Cut Locus. The *point source unfolding* cuts the *cut locus* of the point x: the closure of set of all those points y to which there is more than one shortest path on $\mathcal{P}$ from x. Our method also relies on the cut locus, but now the cut locus $C_P(Q)$ on the half-surface P with respect to its boundary $\partial P = Q$. As cut loci in this paper will always be with respect to Q, we will simplify the notation to C_P. To obtain the unfolding of $\mathcal{P}$, we cut most edges of the cut loci, and some additional cuts that will be explained shortly.

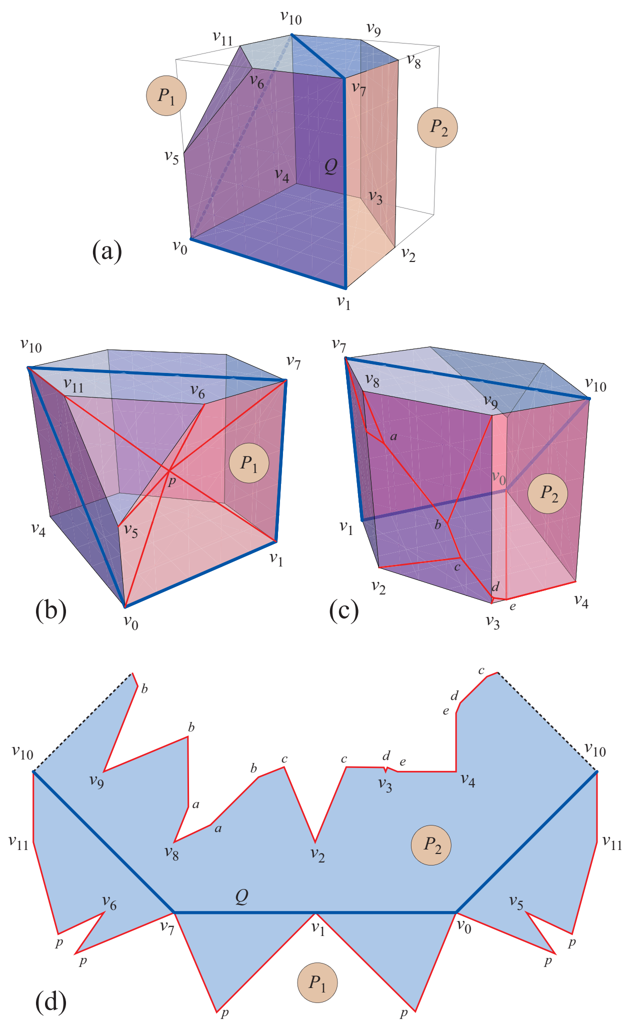


Figure 1: (a) Truncated cube and quasigeodesic $Q = (v_0, v_1, v_7, v_{10})$. (b) View of P_1 and C_{P_1}. (c) View of P_2 and C_{P_2}. (d) Unfolding to a simple polygon.

Convex Curves, Slice Curves, and Quasigeodesics. Let $p \in Q$ be a point on an oriented, simple closed polygonal curve Q on $\mathcal{P}$. Let $L(p)$ be the total face angle incident to the left side of p, and $R(p)$ the angle to the right side. Q is a *convex curve* if $L(p) \leq \pi$ for all points of Q. Q is a *quasigeodesic* if $L(p) \leq \pi$ and $R(p) \leq \pi$ for all p, i.e., it is convex to both

[*]Dept. Math., Faculty Educ., Kumamoto Univ., Kumamoto 860-8555, Japan. j-itoh@kumamoto-u.ac.jp

[†]Dept. Comput. Sci., Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu.

[‡]Inst. Math. 'Simion Stoilow' Romanian Acad., P.O. Box 1-764, RO-014700 Bucharest, Romania. Costin.Vilcu@imar.ro.

sides. A quasigeodesic loop has a single exceptional point at which the angle conditions may not hold. We have the inclusions: (convex) $\supset$ (quasigeodesic loop) $\supset$ (quasigeodesic). Q is a *slice curve* if it is the intersection of $\mathcal{P}$ with a plane. It was known that both convex curves [OS89] and slice curves [O'R03] are *simply developable*: they "develop" on the plane without self-intersections. We establish elsewhere [OV09] that both convex and slice curves "live on a cone," which both explains why they are simply developable and ensures that the source unfoldings we describe here applies.

Example. Throughout we will use an example polyhedron $\mathcal{P}$ to illustrate concepts: a cube twice truncated, with Q the particular closed quasigeodesic shown in Fig. 1(a). The angles at the vertices of $Q = (v_0, v_1, v_7, v_{10})$ within P_1 are, respectively, $(\frac{3}{4}\pi, \frac{1}{2}\pi, \frac{3}{4}\pi, \frac{1}{2}\pi)$, and the angles at those vertices within P_2 are, $(\frac{3}{4}\pi, \pi, \frac{3}{4}\pi, \pi)$. Because all of these angles are $\leq \pi$, Q is convex to both sides and so a quasigeodesic. For Q a simple closed quasigeodesic, the cut loci C_{P_i} are each a single tree with each edge a (geodesic) segment. For more general Q, edges of the cut locus can also be parabolic arcs, and the cut locus consists of a forest on one of the two sides.

2 Overview of Method

We first provide an overview of the method, illustrating the steps through the example in Fig. 1.

Step 1. We show that, for Q in the class identified, a small neighborhood of Q within one of the halves P lives on a cone P^*, which then may be flattened to a doubly-covered cone in the plane. Figures 2 and 3 show natural flattening choices for P_1^* and P_2^* among the many available. In these figures, both Q and the faces of P_i incident to Q are shown. Let $C_{P_i^*}$ be the cut locus of Q on P_i^*, also shown in the figure.

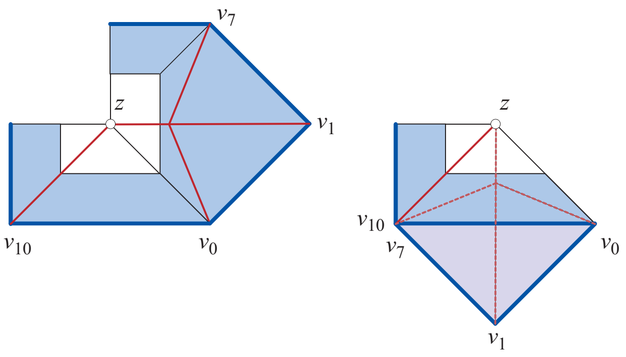


Figure 2: Doubly-covered cone P_1^* and the cut locus $C_{P_1^*}$. (a) Cone opened, showing front and back side-by-side. (b) Both sides overlaid, creased through v_0. Dashed edges are on the back. Angle at z is $\gamma = \frac{3}{4}\pi$.

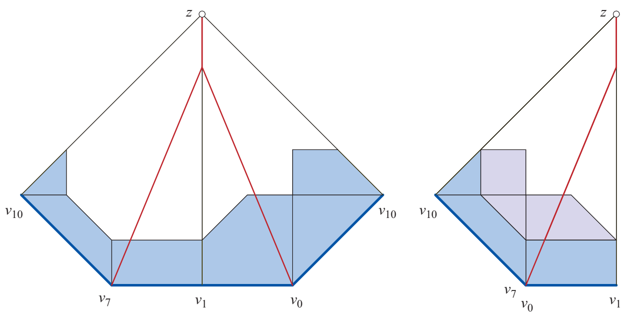


Figure 3: Doubly-covered cone P_2^* and the cut locus $C_{P_2^*}$. (a) Cone opened, showing front and back side-by-side. (b) Both sides overlaid, creased through v_1. Angle at z is $\gamma = \frac{1}{4}\pi$.

Step 2. Next we define a notion of *peels* and *subpeels*, both for P and for P^*, for each half P. The surface is partitioned into peels bounded by Q and the cut locus, and each peel is partitioned into subpeels. We defer the technical definition to Section 5, and for now rely on Fig. 4, which displays the peels and subpeels of P_1.

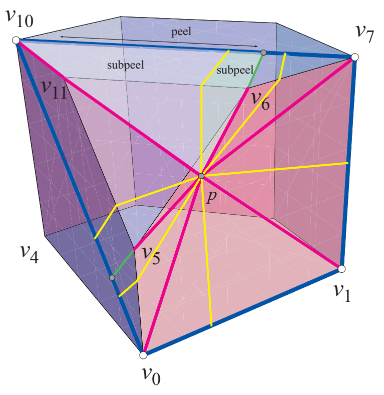


Figure 4: Cut locus C_{P_1} (red); peels between marked points on Q, either leaves of C_{P_1} or projections of leaves (green paths); and subpeels (yellow paths).

A key result (Lemma 2) establishes that the peels of P are nested inside peels of P^*.

Step 3. For each half P, cut C_P along its edges not incident to Q. By our peel nesting lemma, this permits embedding the cut version of P into cone P^*. Now we cut an edge of the flattened P^*, and open it, as illustrated in Fig. 6(a,b). This leads to our first main result (Theorem 4): the curve source unfolding of each half P unfolds without overlap.

Step 4. Knowing that each half unfolds without overlap, we now join the two halves together. How this is accomplished depends on the type of curve Q. Here we only illustrate a method that applies to all convex curves. For the convex side (say, P_2 in our example), we cut as described above: all edges of C_{P_2} not incident to Q, plus a *generator* of the doubly-covered cone to slit it open: $v_{10}z$ in our example. For the nonconvex side (P_1, but in fact both sides are convex in the example), we cut the entire cut locus C_{P_1}, except those edges incident to vertices of Q that are not vertices of $\mathcal{P}$, and some additional cuts. For our example, those additional cuts are empty, and we obtain the unfolding shown in Fig. 1(d). For more general convex Q, for all vertices v of $\mathcal{P}$ in Q that are not incident to C_{P_1}, we make one cut orthogonally from v to C_{P_1}. And finally we cut a corresponding generator of P_1^*. These cuts partition P_1 into domains that project orthogonally onto Q, and so avoid overlap.

The halves determined by slice curves are joined differently, not described in this abstract.

3 Doubly-Covered Cone

Let F be the set of faces of P, and F_Q the subset of F that touches Q: $F_Q = \{f \in F : f \cap Q \neq \varnothing\}$. Cut all the edges of F_Q that are not incident to Q. The result is a surface B_Q that is topologically a "band" or "ring" bounded by two topological circles: Q, and the cut edges. B_Q is locally flat.

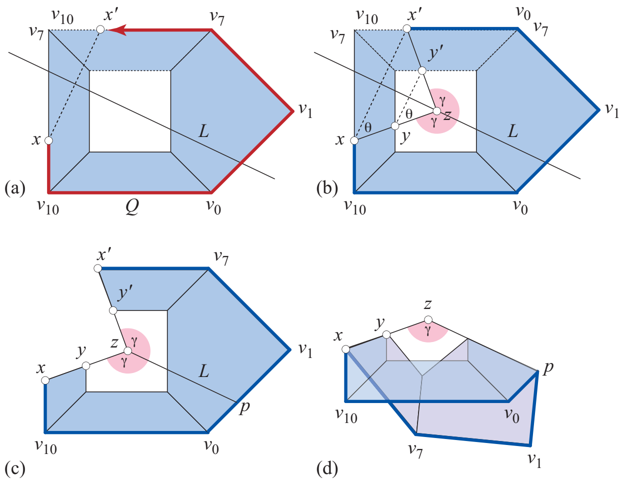


Figure 5: (a) Q is developed from x counterclockwise around to its image x'. (b) $\theta = \angle zxx' = \frac{1}{4}\pi$ leads to $\gamma = \frac{3}{4}\pi$. (c) Cutting B_Q along xy. (d) Reflecting/creasing over L. The x choice here leads to a different embedding than in Fig. 2(b).

Next we describe a procedure for flattening a small neighborhood of Q in B_Q to a subset of a doubly-covered planar cone P^*. P^* will have apex z and two straight sides incident to z. Select a point $x \in Q$ that will ultimately lie on one of these straight sides xz. Develop Q in the plane according to the incident face angles of B_Q from x counterclockwise around Q until the second image x' of x is encountered again. See Fig. 5(a). The apex z is computed by requiring that the curvature at z be $2\pi - \tau_Q$, where τ_Q is the total turn of Q. This forces $\gamma = \frac{1}{2}\tau_Q$ in Fig. 5(b,c,d).

Although not every simply developable Q lies on this cone, we can prove that it does for certain classes:

Lemma 1 *If Q is a simple closed convex curve, or a slice curve, it lives on a cone, and the development of Q lives on the flattened double-cone P^*.*

4 Cut Locus of Q

Fix P, and let $q_0, q_1, \ldots, q_k$ be the vertices of Q in some circular order. A *vertex of* Q is a point q with a nonzero angle gap in P at q. The cut locus C_P is a tree whose leaves span the vertices of P, including the vertices of Q, which must be leaves of C_P. See Figures 1(b,c). On the unbounded side of the cone, the cut locus may be a forest with branches to infinity; but we set this aside in this abstract.

Any point $p \in C_P$ has one or more *projections* to Q: endpoints of shortest paths from p to Q. Non-leaf points p of C_P have at least two distinct projections.

The cone P^* and surface P from which it derives share the same boundary Q, and by construction, the same angle gaps occur along Q. Therefore, Q has the same vertices in both P^* and P. Thus C_P and C_{P^*} both have the same set of leaves touching Q, but of course in general they differ in the interior of the surfaces.

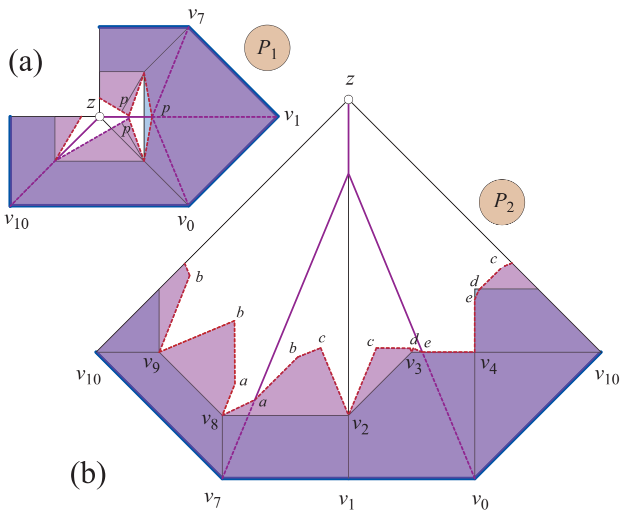


Figure 6: (a) Nesting of cut P_1 within the unfolded P_1^* of Fig. 2(a). (b) Similar nesting of P_2 within P_2^* of Fig. 3(a).

C_{P^*} is determined by any small neighborhood of Q in B_Q, because P^* is so determined. Because every leaf of C_{P^*} is also a leaf of C_P, the bisecting property of cut loci (each edge of the cut locus incident to a vertex $v \in Q$ bisects the angle of P at v) implies that small neighborhoods of the leaves of C_{P^*} are included in C_P. In other words, the edges of C_P and C_{P^*} issuing from vertices of Q coincide until they hit a vertex of C_P or of P. We use this property in the proof of Lemma 2 below.

5 Peels & Subpeels

Let $u_0, u_1, \ldots, u_m$ be the vertices (leaves and junction points) of C_P, following a circular ordering of their projections to Q. Let (u_j, u_k) be two consecutive leaves of C_P, and (u'_j, u'_k) corresponding consecutive projections onto Q. The *peel* $\alpha_P(u_j, u_k)$ is the closed flat region of P bounded by the two projection paths $u_j u'_j$, $u_k u'_k$, the subpath of Q from u'_j to u'_k, and the unique path in C_P connecting u_j to u_k. Each peel is isometric to a planar convex polygon.

Between two consecutive leaves u_j and u_k are the vertices along the tree path, $u_j, u_{j+1}, \ldots, u_{k-1}, u_k$. Each of these delimits a *subpeel* of $\alpha_P(u_j, u_k)$, partitioning the peel along the projection segments $u_l u'_l$, $j < l < k$. See Fig. 4.

The notion of peel and subpeel can be defined exactly analogously for P^*. We will use α for peels and β for subpeels.

Lemma 2 (Peel Nesting) *Each subpeel β_P of P is isometric to a region of a subpeel β_{P^*} of P^*. The union of the subpeels in one peel α_P of P is non-overlapping, and thus each peel α_P is nested inside a peel α_{P^*} of P^*.*

Proof. *Idea.* Let Γ be the directed curve that traces on P^* a maximal subtree of C_P disjoint from Q. We let $x_0 \in \Gamma$ be any point in the interior of a peel α_{P^*}. Now we move a point x_t along Γ continuously from x_0. As long as x_t remains inside the peel α_{P^*}, the subpeel β_P to which x_t belongs remains included in peel α_{P^*}. Now assume that x_t reaches a point of $x_t{=}y \in C_{P^*}$ and crosses C^*_P there, as it does at $y{=}a$ in Fig. 7. This is exactly when the claim of the lemma might be FALSE, for the subpeel β_P bounded by Γ then extends beyond the peel α_{P^*}. Using a comparison theorem of Alexandrov, we show that y must be a junction point of C_P. This means that the subpeel β_P ends at y, and Γ enters a new subpeel β'_P of a new peel α'_{P^*} of P^*. Thus the nesting claim of the lemma is established. □

6 Source Unfolding

The *half-polyhedron source unfolding* of Q is obtained by cutting the edges of C_P not incident to Q, embedding the peels within P^* via Lemma 2, and cutting a generator xz of the doubly-covered cone and opening.

Theorem 3 (Half-Polyhedron Source Unfolding) *For any Q for which the conclusion of Lemma 1 holds, the source unfolding of each half P of $\mathcal{P}$ is non-overlapping.*

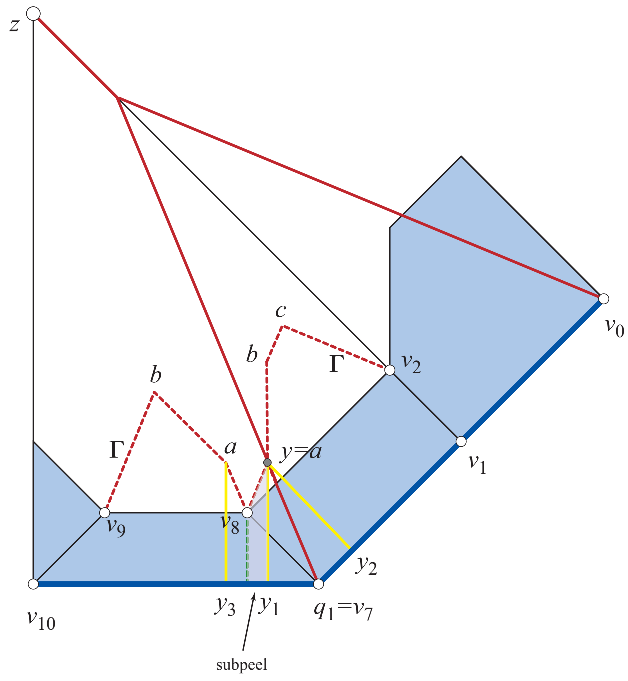


Figure 7: Γ is the dashed curve, directed from v_9 to v_8. x_t crosses C_{P^*} at y, which coincides with the junction point a of C_P. Γ enters the new subpeel $\beta_{P^*}(v_7, y_2)$ and the new peel is $\alpha_{P^*}(v_7, v_0)$.

Several strategies are available for joining the two halves, one of which is illustrated in Fig. 1(d).

Theorem 4 (Full Source Unfolding) *For Q a convex curve, or a slice curve containing at most one vertex, additional cuts permit joining the halves to one, simple polygon.*

Finally, we note that for a convex curve Q shrinking to a point x, the full source unfolding w.r.t. Q approaches the point source unfolding w.r.t x.

References

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, July 2007. `http://www.gfalop.org`.

[IOV08] Jin-ichi Itoh, Joseph O'Rourke, and Costin Vîlcu. Unfolding convex polyhedra via quasi-geodesic star unfoldings. Technical Report 091, Smith College, December 2008. arXiv:0821.2257v1 [cs.CG].

[O'R03] Joseph O'Rourke. On the development of the intersection of a plane with a polytope. *Comput. Geom. Theory Appl.*, 24(1):3–10, 2003.

[OS89] Joseph O'Rourke and Catherine Schevon. On the development of closed convex curves on 3-polytopes. *J. Geom.*, 13:152–157, 1989.

[OV09] Joseph O'Rourke and Costin Vîlcu. Conical existence of certain closed curves on polyhedra. Manuscript in preparation, 2009.

Locked Thick Chains

Erik D. Demaine* Martin L. Demaine* Stefan Langerman†‡ Jérôme Vervier‡

Abstract

We investigate when *thick* 3D polygonal chains are *locked*, i.e., have a disconnected configuration space. In particular, we show that thick 4-chains are never locked, and we exhibit a class of locked thick 5-chains whose ratio of maximum edge length to minimum edge length is strictly less than 3 (the best known ratio for nonthick chains).

1 Introduction

In the study of linkage folding (see, e.g., [5, Part I]), the typical mathematical model of a mechanical linkage is a collection of rigid line segments (*bars*) permanently attached at certain endpoints (*joints*). Although there are many types of joints, the most common mathematical model is a *universal* joint, which can flex arbitrarily. Bars are usually considered physical objects that cannot intersect each other (usually in 3D). Thus a linkage can move according to any continuous motion such that the bar lengths remain preserved, the bars remain attached according to the joints, and the bars do not intersect each other.

In this paper, we consider changing this model to allow the bars to have positive thickness. Our motivation is that such a model is physically more realistic, in particular when modeling physical objects such as mechanical linkages or protein backbones. O'Rourke [5, Sect. 6.3.3, p. 91] introduced one model of thick 3D linkages, where the chain is a Minkowski sum of a regular nonthick 3D linkage and a ball, turning edges into cylinders and vertices into identical spheres. We distinguish the thick edges and thick joints of the resulting shape from the original nonthick bars and nonthick joints. The linkage is considered *non-self-intersecting* if the only thick bars that intersect each other are those whose underlying nonthick edges share a nonthick vertex. A slight variation is easier to build in practice, and even exists in many magnetic ball/rod construction kits: if two thick edges intersect, they should be from incident nonthick bars *and* the thick edges should intersect only within their shared thick joint. So far no nontrivial theorems have been established in either model.

This paper considers thick "locked chains" in O'Rourke's model and our variation thereof. A *chain* is a linkage in which the bars form a polygonal chain, connected in a path configuration. A linkage is *locked* if its configuration is disconnected, that is, there are two configurations that cannot reach each other by continuous (non-self-intersecting) motions.

O'Rourke originally introduced thick linkages with the following open problem: is there a locked thick chain whose underlying nonthick bars have unit length? [5, Open Prob. 6.1, p. 91] This problem remains open. It is motivated by the analogous problem for nonthick chains [1, 5, Open Prob. 9.19, p. 151], which also remains open. A more general version of the problem was posed by Demaine and O'Rourke [5, Open Prob. 9.20, p. 152]: what is the smallest possible length ratio between the longest bar and the shortest bar that admits a locked nonthick chain? The smallest ratio known is the original five-bar chain which achieves a ratio of $3+\varepsilon$ [2, 1, 5, Thm. 6.3.1, p. 89]. The case of unit-length bars is the smallest possible ratio of 1. But so far nothing better than 3 is known, for nonthick or thick chains.

We analyze a family of locked thick chains that achieve a length ratio of strictly less than 3. In fact, the family is parameterized by the bar thickness as a multiple λ of the minimum bar length. Whenever a chain with the necessary structural properties exists, we prove that the chain is locked when its bar-length ratio is at least

$$\sqrt{9-8\cdot\left(\frac{\lambda}{1-\lambda}\right)^2}$$

which is strictly less than 3 for all $\lambda > 0$.

Our locked thick chain has five bars, the same as the classic example of a locked nonthick chain. Indeed, it is known that all locked nonthick chains have at least five bars [2]. As a complementary result, we prove that all locked thick chains have at least five bars as well.

2 Preliminaries

For both thick and nonthick chains, we distinguish the chain (sequence of bars moving in space) from its configuration (a chain in a particular position). A polygonal chain P in $\mathbb{R}^d$ is a sequence of fixed length bars connected at their successive endpoints and moving freely in a d-dimensional space. The chain has $n+1$ vertices $V = \langle v_0, \ldots, v_n\rangle$, and is specified by the fixed edge lengths d_i between v_i and v_{i+1}, $i = 0, \ldots, n-1$. We write $P[i,j]$, $i \le j$, for the polygonal subchain composed of vertices $v_i, \ldots, v_j$. The *joints* correspond to the internal vertices $v_1, \ldots, v_{n-1}$.

A *configuration* $Q = \langle q_0, \ldots, q_n\rangle$ of the chain P is an embedding of P into $\mathbb{R}^d$, i.e., a mapping of each vertex v_i to a point $q_i \in \mathbb{R}^d$, satisfying the constraints that the distance between q_i and q_{i+1} is d_i. The points q_i and q_{i+1} are connected by a straight line segment e_i.

Let B^d_λ be the ball in $\mathbb{R}^d$ of radius λ centered at the origin, and let e be a straight line segment in $\mathbb{R}^d$. The *thick bar* of thickness λ and of skeleton e is the Minkowski sum $e \oplus B^d_\lambda$ between e and B^d_λ. A *thick chain* P_λ in $\mathbb{R}^d$ is

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139, USA. {edemaine,mdemaine}@mit.edu

†Maître de recherches du F.R.S.-FNRS

‡Computer Science Department, Université Libre de Bruxelles, CP 212, Bvd. du Triomphe, 1050 Brussels, Belgium. stefan.langerman@ulb.ac.be, j.vervier@gmail.com

specified by its skeleton P and its thickness λ. It can be seen as a sequence of thick bars whose end balls coincide. A configuration Q_λ of a thick chain P_λ is the Minkowski sum between a configuration Q of P and the ball B^d_λ. The corresponding configuration Q of P is called the *skeleton* of configuration Q_λ. Note that adjacent thick bars of a thick chain always intersect. We however impose that a configuration be *simple*, that is, nonadjacent bars do not intersect. Furthermore, we require that the thick bar $e_{i-1} \oplus B^d_\lambda$ does not intersect the ball $q_{i+1} \oplus B^d_\lambda$. A motion of a chain is simple if every configuration during the motion is simple.

An *expansive motion* of a chain P is a motion with the property that the distance between any pair of points on the chain monotonically increases with time [3]. We say that the motion of a thick chain is *expansive* if the motion of its skeleton is expansive. We can then show the following:

Lemma 1 *An expansive motion of a thick chain starting from a simple configuration, is simple.*

3 Thick 4-Chains Cannot Lock

It is known that a nonthick 4-chain cannot be locked [2]. Following an idea similar to that in [4], we use a linear transformation as a first step toward bringing the skeleton of the thick chain in 2D. Define the parameterized linear transformation $f_\tau : \mathbb{R}^3 \to \mathbb{R}^3$ by $f_\tau(x, y, z) := (\tau x, y, z)$ with parameter $\tau \geq 1$. For a set $S \subseteq \mathbb{R}^3$, write $f_\tau(S) = \{f_\tau(p) : p \in S\}$. Note that linear functions are distributive over the Minkowski sum: that is, for all sets A and B,

$$f_\tau(A \oplus B) = f_\tau(A) \oplus f_\tau(B). \tag{1}$$

Further, because $\tau \geq 1$, we have

$$B^3_\lambda \subseteq f_\tau(B^3_\lambda). \tag{2}$$

Theorem 2 *Every simple thick* 4*-chain can be straightened in 3D.*

Proof. Let $C_\lambda = C \oplus B^3_\lambda$ be a thick 4-chain of thickness λ of skeleton C whose vertices are $(v_0, v_1, v_2, v_3, v_4)$.

Consider the plane K formed by the two middle nonthick bars of the skeleton, $e_1 = (v_1, v_2)$ and $e_2 = (v_2, v_3)$. We may choose the coordinate system so that K is the yz plane. We can also apply small perturbations to v_0 and v_4 to ensure that they are not in K. Let k_1 and k_2 be the first and last (nonthick) bars of C, that is, $k_1 = (v_0, v_1)$ and $k_2 = (v_3, v_4)$.

Definition the parameterized linear transformation f_τ as above. The two middle bars, e_1 and e_2, have the same length in $f_\tau(C)$ as in C, but the transformation increases the length of the two end bars, k_1 and k_2. So we have to truncate the length of k_1 and k_2 to preserve their length. Also, the thick bars of C_λ, which are cylinders with a sphere at each end, become cylinders with a ellipsoid basis in $f_\tau(C_\lambda)$. In order to obtain a thick chain again, the basis of this cylinder has to be changed to a disk.

By truncating the length of the end bars, we obtain a new 4-chain C^τ from $f_\tau(C)$. Clearly, $C^\tau \subseteq f_\tau(C)$. By equation (2), $B^3_\lambda \subseteq f(B^3_\lambda)$. So,

$$C^\tau_\lambda = C^\tau \oplus B^3_\lambda \subseteq f_\tau(C) \oplus f_\tau(B^3_\lambda) = f_\tau(C \oplus B^3_\lambda) = f_\tau(C_\lambda).$$

We want to apply the linear motion f_τ for τ ranging from 1 to ∞. To achieve this motion in finite time, we define another motion parameterized by t from 0 to 1 that applies f_τ where $\tau = 1/(1-t)$. Throughout the motion, we truncate the length of the two exterior segments and the basis of the ellipsoid cylinder. Because the linear transformations preserve intersection among regions, two thick bars that do not intersect before the motion do not intersect during the motion. As t approaches 1, τ grows to infinity and the exterior thick bars become perpendicular to the yz plane. Let C' be that skeleton of the chain at that point in time.

Two possible situations arise: either the vertices v_0 and v_4 are on the same side of the yz plane or they lie on opposite sides of the yz plane; see Fig. 1 (a–c) for top and side views. Let Π_1 be the plane containing v_0, v_1, and v_2, and Π_2 be the plane containing v_2, v_3, and v_4. Let ℓ be the line at the intersection of Π_1 and Π_2. Notice that ℓ intersects C' only at vertex v_2. Thus the subchain $v_0v_1v_2$ is contained in a halfplane of Π_1 bounded by ℓ and the subchain $v_2v_3v_4$ is contained in a halfplane of Π_2 bounded by ℓ. By hinging those two halfplanes about ℓ, we obtain an expansive motion of C' that makes it planar. By Lemma 1, this motion is simple for the thick chain.

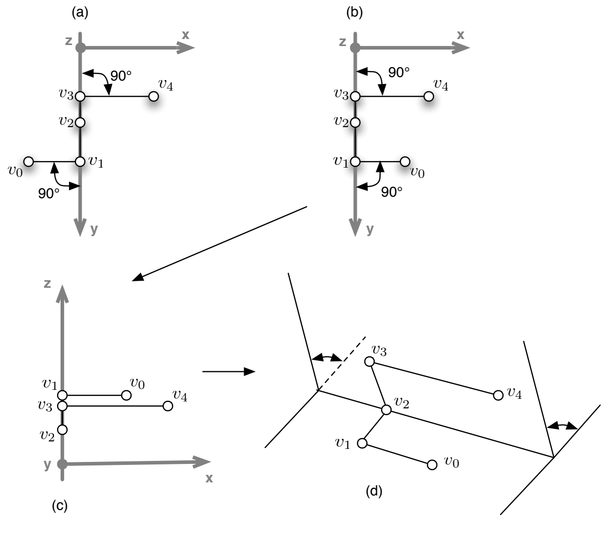


Figure 1: 4-chain: Example where we bring back the two exterior thick bars in the plan yz.

Finally, we straighten the resulting planar 4-chain using an expansive motion [3]. Again, by Lemma 1, this motion is simple for the thick chain. $\square$

4 Locked Thick 5-Chains

4.1 Introduction

Consider a 5-chain of thickness λ with end bars (v_0, v_1) and (v_4, v_5) of length $s > 1$ and middle-bars (v_1, v_2), (v_2, v_3) and (v_3, v_4) of length 1. Because the chain is simple, $\lambda < 1/2$. We will determine bounds on s so that the chain can be locked.

Fig. 2 shows the orthogonal projection of two adjacent thick bars (thickness λ) and one more thick bar perpendicular to the other two. The projection is over to the plane formed by the skeleton of the two adjacent thick bars. We first bound the distance between the points C and v_0 depending on the angle α between the two adjacent thick bars and on the thickness λ.

Lemma 3 *Consider in 2D two adjacent thick bars (v_0, v_1) and (v_0, v_2) of thickness λ, consider a circle of radius λ centered at the point C which is in the interior of the angle α formed by the two adjacent thick bars and which does not intersect any of these two thick bars (see Fig. 2). Let d be the distance between v_0 and C. Then*

$$d \geq \frac{2 \cdot \lambda}{\sin(\alpha/2)}.$$

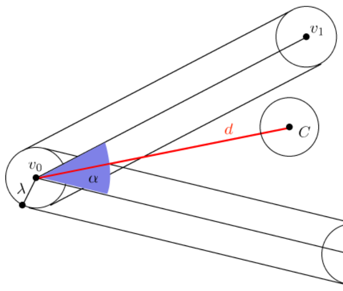


Figure 2: Lemma 3.

Lemma 4 *Let P_λ be a 5-chain of thickness λ (see Fig. 3) which is the Minkowski Sum of a nonthick 5-chain P of vertices $(v_0, v_1, v_2, v_3, v_4, v_5)$ with the ball B^3_λ. If the three middle bars are of length one, and the central projection from v_4 on the plane $v_1v_2v_3$ is as in Fig. 4, then the distance s between the middle point of the three middle thick bars of P and the vertex v_1 (or the vertex v_4) is smaller than*

$$\frac{1}{2}\sqrt{9 - 8 \cdot \left(\frac{\lambda}{1-\lambda}\right)^2}.$$

Proof. Consider the central projection of P_λ from v_4 on plane K containing v_1, v_2, and v_3 (see Fig. 4). The thick bar v_4v_5 intersects K in an ellipsoid centered at the point D. This ellipsoid contains a circle of radius λ centered at D. Thus replacing the ellipsoid by a circle centered at D is a weaker constraint on the motion. Applying Lemma 3, we bound the distance d between v_1 and D:

$$d \geq \frac{2 \cdot \lambda}{\sin(\alpha/2)}$$

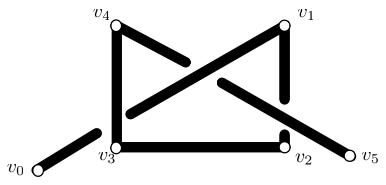


Figure 3: Thick 5-chain.

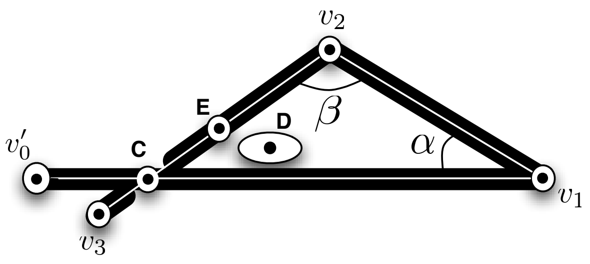


Figure 4: Central projection of the thick 5-chain from v_4 on the plane K containing v_1, v_2, and v_3. The thick bars v_1v_2, v_2v_3 and v_0v_1 are shown, as well as the ellipsoid formed by the thick bar v_4v_5 as it intersects plane K.

Because the projection is on K, the length of thick bars v_1v_2 and v_2v_3 are preserved and their thickness can only be larger. Let v_0' be the projection of v_0 on K and C be the intersection in the projection between $v_0'v_1$ and v_2v_3. By assumption, this intersection always exists. Consider the triangle v_1v_2C. Since $|v_2v_3| = 1$, $|v_2C| \leq 1 - 2\lambda$. Also $|v_1v_2| = 1$ and by the triangle inequality, $d = |v_1C| \leq 2 - 2\lambda$.

Because the central projection is from v_4 over K, $|v_0'v_1| > |v_0v_1|$.

$$2 - 2 \cdot \lambda > d \geq \frac{2 \cdot \lambda}{\sin(\alpha/2)} \qquad (3)$$

We have to bound the angle α by the equation (3):

$$\begin{aligned} 2 - 2 \cdot \lambda &> \frac{2 \cdot \lambda}{\sin(\alpha/2)} \\ \sin(\alpha/2) &> \frac{\lambda}{1-\lambda} \\ \frac{\alpha}{2} &> \arcsin\frac{\lambda}{1-\lambda} \\ \alpha &> 2\arcsin\frac{\lambda}{1-\lambda} \end{aligned}$$

Because, in every triangle, the sum of angles is equal to π, we have β to $\beta < \pi - \alpha$.

With the angles α lower-bounded and β upper-bounded, we have enough information to bound the distance between the middle point of the three middle thick bars of P and the point A (the vertex v_1).

Let be $L = l_2 + l_3 + l_4 = 3$ the total length of the three middle thick bars and E the middle point of the three middle thick bars: the distance between E and the vertices v_4 and v_1 is equal to $3/2$. By the length attributed to the three middle thick bars, the point E is at the center of the thick bar v_2v_3 (see Fig. 4).

Consider the triangle EBA (obtained by considering the point E in Fig. 4) to find the distance between the points E and A.

In this new triangle, we know : the angle β ($\beta < \pi - \alpha$), the length $|AB|$ (by the definition of P, $|AB| = 1$) and the length $|EB|$ (by the definition of P and the middle point E, $|EB| = \frac{1}{2}$).

By Al-Kashi's theorem, we have $|EA| = \sqrt{|AB|^2 + |EB|^2 - 2 \cdot |AB| \cdot |EB| \cos\beta}$. $|EA|$ corresponds to the distance between the middle point E of the three thick bars of P and the vertex v_1 so $|EA| = \sqrt{1 + \frac{1}{4} - 2 \cdot 1 \cdot \frac{1}{2}\cos\beta}$. This distance is the same if we consider the vertex v_4 instead of the vertex v_1 because the three middle thick bars have length one.

So $s = |EA| = \sqrt{1.25 - \cos\beta}$:

$$\begin{array}{lrcl}
\text{As} & \beta & < & \pi - \alpha \\
\text{So} & \cos(\beta) & > & \cos(\pi - \alpha) = -\cos\alpha \\
\text{And so} & s & < & \sqrt{1.25 + \cos\alpha} \\
\text{As} & \alpha & > & 2 \arcsin \frac{\lambda}{1-\lambda} \\
\text{Then} & \cos\alpha & < & \cos(2 \arcsin \frac{\lambda}{1-\lambda}) \\
\text{But} & \cos(2 \arcsin \frac{\lambda}{1-\lambda}) & = & \cos^2 \arcsin \frac{\lambda}{1-\lambda} - \sin^2 \arcsin \frac{\lambda}{1-\lambda} \\
 & & = & 1 - 2 \sin^2 \arcsin \frac{\lambda}{1-\lambda} \\
 & & = & 1 - 2\left(\frac{\lambda}{1-\lambda}\right)^2 \\
\text{Then} & \cos\alpha & < & 1 - 2(\frac{\lambda}{1-\lambda})^2 \\
\text{And so} & s & < & \sqrt{1.25 + 1 - 2\left(\frac{\lambda}{1-\lambda}\right)^2} \\
 & & < & \sqrt{2.25 - 2(\frac{\lambda}{1-\lambda})^2}
\end{array}$$

□

This lemma bounds distances from the middle point E of the three middle bars of the thick 5-chain. We will re-use the demonstration of [5] but with this distances as radius of the centered sphere.

Theorem 5 *Let K be a 5-chain of thickness λ (see Fig. 3), result of the Minkowski Sum between a nonthick 5-chain of vertices $(v_0, v_1, v_2, v_3, v_4, v_5)$ and the ball B^3_λ. If the length of each of the three middle thick bars is set to one and the end bars are of length greater than $\sqrt{9 - 8(\frac{\lambda}{1-\lambda})^2}$ then K is locked.*

Proof. Let $D = \frac{1}{2}\sqrt{9 - 8 \cdot (\frac{\lambda}{1-\lambda})^2}$ be the upper-bound on the distance between the middle point E of the three middle thick bars of K and the vertex v_1 (or v_4) given by the lemma 4. Let $r = D + \varepsilon$, for $\varepsilon > 0$ small and the sphere F of radius r centered at E. By construction, the vertices $\{v_1, v_2, v_3, v_4\}$ are in F for all reconfiguration of K.

We fix l_1 and l_5 to at least $2 \cdot r + \varepsilon = 2 \cdot D + 3 \cdot \varepsilon$.

Because l_1 and l_5 are greater than the diameter of F, the vertices v_0 et v_5 are not in F for all reconfiguration of K.

We first claim that unless v_0 or v_5 enter F, the projection from v_4 on the plane through $v_1v_2v_3$ is as in Fig. 4, that is, the projection of v_0v_1 intersects v_2v_3. Symmetrically, we claim that in the projection from v_1 on the plane containing $v_2v_3v_4$, the projection of v_4v_5 intersects v_2v_3. Suppose that the former claim gets violated first during the motion. In that case, the projection of v_0 has to lie on v_2v_3 but in that case, v_0 is inside the triangle $v_2v_3v_4$ which itself is inside F, a contradiction.

Assume by contradiction that there is an unlocking motion for K.

If we close K by adding a string along the surface of F between its two free ends, then we obtain a trefoil knot.

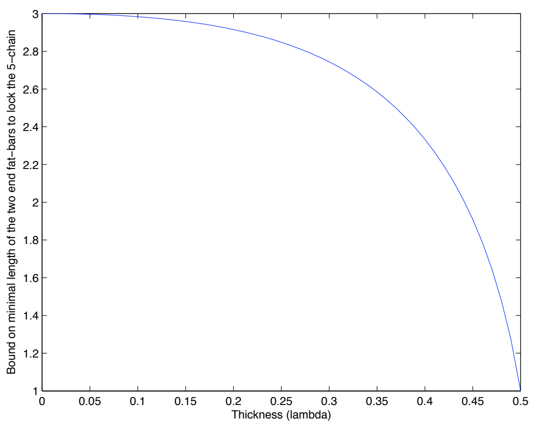


Figure 5: Bounds from Theorem 5

Because F separates (by its boundary) the two sets $\{v_0, v_5\}$ and $\{v_1, v_2, v_3, v_4\}$, we can attach a sufficiently long unknotted string s from v_0 to v_5 exterior to F. We obtain a trefoil knot.

By our assumption, we can, by an existing motion, unlock K (note that the topology of a knot does not change during a motion), at the end of this motion, we obtain a unlocked knot (the trivial knot). This result is in contradiction with the fact that $K \cup s$ is a trefoil knot when we add to it a string s, then K cannot be unlock by any motion. □

We have first bound on the minimal length of the two end bars needed to lock a thick 5-chain of thickness λ. A first plot may be drawn using this formula (see Fig. 5). In this plot, the x axis represent the thickness (λ) and the y axis our bound on the minimal length to assign to the two end bars of the thick 5-chain.

Interestingly, if we put the thickness parameter λ to zero (we are so in the case of a nonthick 5-chain) then we obtain the same result than [5].

References

[1] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. Locked and unlocked polygonal chains in three dimensions. *Discrete & Computational Geometry*, 26(3):269–281, October 2001. Full version is arXiv:cs.CG/9910009.

[2] J. Cantarella and H. Johnston. Nontrivial embeddings of polygonal intervals and unknots in 3-space. *Journal of Knot Theory and Its Ramifications*, 7(8):1027–1039, 1998.

[3] R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(2):205–239, September 2003.

[4] E. D. Demaine, S. Langerman, J. O'Rourke, and J. Snoeyink. Interlocked open linkages with few joints. In *Proceedings of the 18th Annual Symposium on Computational Geometry*, pages 189–198, 2002.

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

Fitting a Point Set by Small Monotone Orthogonal Chains

J.M. Díaz-Báñez[*] M. A. Lopez[†] C. Seara[‡] I. Ventura[§]

Abstract

We study the problem of fitting a monotone orthogonal chain with a small number of links to a set S of n points in the plane. The objective function is the maximum orthogonal distance from S to the chain. We show that the problem of fitting a chain with 2 links can be solved in $O(n)$ time if the orientation is fixed and $O(n \log n)$ time when the orientation is not a priori known. Both algorithms are optimal.

1 Introduction and preliminaries

Fitting a curve of a certain type to a given point set in the plane is a fundamental problem with applications in many fields. In the *Min-Max problem* a polygonal chain with k corners or joints is fitted to a data set with the goal of minimizing the maximum vertical distance from the input points to the chain (the so called *Chebyshev error*). This problem was first posed in [6] and solved in $O(n^2 \log n)$ time. The complexity has since been improved, first to $O(n^2)$ time [11] and then to $O(n \log n)$ time [5].

We consider the case in which the points are fitted by a *monotone orthogonal polygonal chain*, i.e., a chain of consecutive orthogonal line segments where the extreme segments are half-lines with the same slope. The case in which the slope is given was first solved in $O(n^2 \log n)$ time [3], and subsequently in $O(n^2)$ [10], and $\min\{n^2, nk \log n\}$ time [8]. Very recently, Fournier and Vigneron [4] give an $O(n)$ time algorithm if the points are sorted according to their x-coordinates, and an $O(n \log n)$ time algorithm for the unsorted case. For the unsorted case the authors prove the optimality for $k = n$, and give an $\Omega(n \log k)$ lower bound for the decision problem. Thus it may be possible to find an algorithm with improved running time depending on k. See Chen and Wang [2] for recent results on some variants of this problem including NP-hard results in 3D.

We focus here on the case where k is small, $k = 2$, and also on the problem of finding the best direction for fitting a monotone orthogonal polygonal chain.

Let $S = \{p_1, \ldots, p_n\}$ be a point set in the plane in general position, where $p_i = (x_i, y_i)$. A *k-orthogonal chain* $\mathcal{O}$, $k \geq 1$, is a chain with $2k - 1$ consecutive orthogonal segments (links) where the extreme segments are half-lines with the same slope. The *orientation* θ of the chain $\mathcal{O}$ is the angle formed by their extreme half-lines with the x-axis. Thus, $\mathcal{O}$ consists of k segments of slope $\tan(\theta)$ and $k-1$ segments of slope $\tan(\theta + \frac{\pi}{2})$. $\mathcal{O}$ is monotone with respect to an orientation α if every line with slope $\tan(\alpha + \frac{\pi}{2})$ intersects the chain either in a point or in a segment with slope $\tan(\alpha + \frac{\pi}{2})$ (Figure 1).

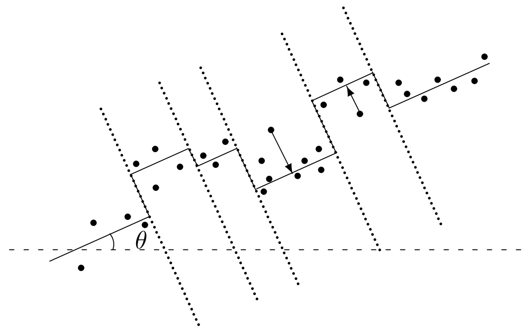


Figure 1: A 6-orthogonal chain.

In this paper we deal with the problem of fitting a monotone k-orthogonal chain $\mathcal{O}$ to a set S of n points in the plane. The chain $\mathcal{O}$ with orientation θ divides the plane into k strips with orientation $(\theta + \frac{\pi}{2})$. Fitting $\mathcal{O}$ to S is to locate k θ-oriented segments $s_i(\theta)$, $i \in \{1 \ldots, k\}$ according to a given optimization criterion.

Let $l_i(\theta)$ be the line passing through $p_i \in S$ with orientation $\theta + \frac{\pi}{2}$. The fitting distance from p_i to $\mathcal{O}$ is given by $d_f(p_i, \mathcal{O}) = \min_{p \in l_i(\theta) \cap \mathcal{O}} d(p_i, p)$. Notice that d_f is not the Euclidean distance. However, d_f is the Euclidean distance between p_i and a point on a segment with orientation θ. The *error tolerance* of $\mathcal{O}$ with respect to S, denoted by $\mu(\mathcal{O}, S)$, is the maximum fitting distance between the points of S and $\mathcal{O}$, i.e., $\mu(\mathcal{O}, S) = \max_{p_i \in S} d_f(p_i, \mathcal{O})$. The *$k$-fitting problem* can now be stated as follows:

Definition 1 *Let S be a set of n points in the plane. The k-fitting problem consists on finding a monotone k-orthogonal chain such that its error tolerance with respect to S is minimized.*

[*]Dept. Matemática Aplicada II, Universidad de Sevilla, Spain, `dbanez@us.es`. Supported by grant MTM2006-03909.

[†]Departament of Mathematics, University of Denver, USA, `mlopez@du.edu`. Supported by grant 90YD0251/01.

[‡]Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain, `carlos.seara@upc.edu`. Supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

[§]Dept. Matemática Aplicada II, Universidad de Sevilla, `iventura@us.es`. Supported by grant MEC MTM2006-03909.

2 The oriented fitting problem

We consider the problem of fitting a k-orthogonal monotone chain with orientation θ. We assume that $\theta = 0$. Thus, we are looking for an x-monotone rectilinear path consisting of an alternating sequence of k horizontal and $k-1$ vertical segments. If the points are unsorted, Fournier and Vigneron [4] give an $O(n \log n)$ time algorithm for this problem which is optimal if $k = n$. The algorithm from Lopez and Mayster is also $O(n \log n)$ if k is a constant. For the sorted case Fournier and Vigneron [4] present an optimal $O(n)$ time algorithm and an $\Omega(n \log k)$ lower bound for the decision problem. Thus, here we consider the problem for small values of k, (in fact $k = 2$), for the unsorted case.

For simplicity we assume that no two points lie on the same vertical or horizontal line. The values $y_{\max} = \max\{y_1, \ldots, y_n\}$ and $y_{\min} = \min\{y_1, \ldots, y_n\}$ (similarly $x_{\max}$ and $x_{\min}$) can be computed in linear time. We assume that the points lie in the first quadrant and that $x_{\min} = 0$, $y_{\min} = 0$, $x_{\max} = c$, and $y_{\max} = d$.

The oriented 2-fitting problem consists of finding two horizontal half-lines joined by a segment contained in a vertical line ℓ. In an optimal solution, ℓ produces a bipartition of S into two subsets: S_1 (S_2) are the points of S to the left (right) of ℓ. The following lemma is straightforward:

Lemma 1 *Line ℓ separates the two points in S with y-coordinates $y_{\min}$ and $y_{\max}$.*

2.1 Linear time algorithm for the 2-fitting problem

In $O(n)$ time compute the median of the x-coordinates of S. Let ℓ be the vertical line at this median. In $O(n)$ time compute the subset S_1 (resp. S_2) of S to the left (resp. right) of ℓ, where $|S_1| = |S_2| = n/2$. In $O(n)$ time compute the tolerances ϵ_1 of S_1 and ϵ_2 of S_2, and store the two pairs of points giving both tolerances. If $\epsilon_1 = \epsilon_2$, stop the algorithm. If $\epsilon_1 < \epsilon_2$ in $O(n/2)$ time compute the median of S_2, reset ℓ as the vertical line at this median value, compute the subsets S_2' and S_2'' of the bipartition of S_2 produced by the new median, compute the tolerances ϵ_2' and ϵ_2'' of S_2' and S_2''. Compute $\max\{\epsilon_2', \epsilon_2''\}$ to decide the next move of ℓ (left or right). Store the tolerance values and the corresponding witness pairs as a temporary values. Next compute and update the tolerances once we know the next move of ℓ. (If $\epsilon_1 > \epsilon_2$ we proceed in a symmetric way). Compare the new tolerance values and continue recursively translating the line ℓ left or right by computing the new median of a subset with half of the points and updating the new tolerance values (left and right) from the old ones. At all times we have two unions at the extremes and an unknown zone in between containing at most two strips.

Clearly, the time complexity of the algorithm is $T(n) = O(n/2) + O(n/4) + O(n/8) + \cdots = O(n)$. By Lemma 1 the optimal line ℓ will be located between the two points in S with y-coordinates $y_{\min}$ and $y_{\max}$. The algorithm stops when either the tolerance values ϵ_1 and ϵ_2 are equal, or when translating ℓ left and right the bigger tolerance changes from ϵ_1 to ϵ_2 and viceversa. In the last case the solution will be the best of the two. Since the algorithm performs a binary search on a unimodal function, the method is correct. Notice that the solution (position of line ℓ or bipartition of S) is not unique because in an optimal solution some points can belong to S_1 or S_2 without changing the solution. We have the following result.

Theorem 2 *The oriented 2-fitting problem can be solved in $\Theta(n)$ time and space.*

By the $\Omega(n \log k)$ lower bound for the decision problem [4], it is clear that if $k = o(\log n)$ there is no linear time algorithm for the unsorted case. Thus, we raise the open question: *for which range of values of k does there exist a linear time algorithm?*

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

We now show a slower $O(n \log n)$ time algorithm for the same problem which has the advantage that it can be used later in the algorithm for the un-oriented 2-fitting problem.

Given two points $p_i, p_j \in S$, a dominance relation is established: p_i *dominates* p_j ($p_j \prec p_i$), if $x_j \leq x_i$ and $y_j \leq y_i$. The relation $\prec$ is a partial order in S. A point $p_i \in S$ is *maximal* if there does not exists $p_j \in S$ such that $i \neq j$ and $p_i \prec p_j$. The maxima problem consists of finding all the maximal points of S under dominance and can be solved optimally in $\Theta(n \log n)$ time [9]. The solution can be extended with respect to any orientation. We are interested in the maxima for any of the four quadrants.

We compute $y_{\max}$, $y_{\min}$, $x_{\max}$, $x_{\min}$, and translate the points in S to the first quadrant of the coordinate system. We assume that $p_1 = (0, y_1)$, $p_n = (c, y_n)$, $p_i = (x_i, y_{\max})$, and $p_j = (x_j, 0)$ (Figure 2). Suppose that p_i is to the left of p_j. By Lemma 1, the vertical line ℓ lies between p_i and p_j. We do the following:

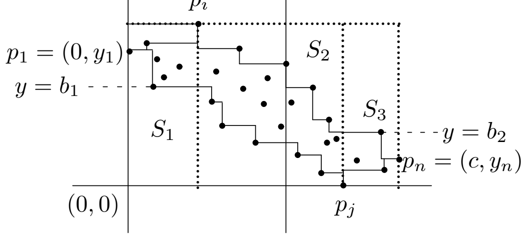


Figure 2: Staircaises formed by maxima points.

1. In linear time classify the points of S as follows: S_1 is the point subset with x-coordinate less than x_i, S_2 is the point subset with x-coordinate in between x_i and x_j, and S_3 is the point subset with x-coordinate greater than x_j. Let n_i be the number of points in S_i, $i = 1, 2, 3$.

2. In $O(n \log n)$ time we can compute the sets of maxima points of S with respect to each of the four quadrants and join them, respectively, forming the staircases structure as in Figure 2.

3. By Lemma 1, the line $\ell := (x = a)$ is in between x_i and x_j, and, in order to find its right location, we can do a *binary search* over the points in the staircase structure (first and third quadrant) in at most $O(\log n_2)$ time such that we get the best balance between the error tolerance on the left and on the right side of ℓ.

It is easy to see that the above staircases structure can be used both for solving the 3-fitting and 4-fitting problems with the same time complexity.

3 The un-oriented 2-fitting problem

The un-oriented 2-fitting problem has the orientation θ as an additional parameter. The goal is to determine an oriented line ℓ_θ with slope $\tan(\pi/2 + \theta)$ which splits S into two subsets S_{l_θ} and S_{r_θ} where $(e^u_{l_\theta}, e^b_{l_\theta})$ and $(e^u_{r_\theta}, e^b_{r_\theta})$ are the pairs of points giving the error tolerance of S_{l_θ} and S_{r_θ}, respectively, and such that its corresponding 2-orthogonal polygonal chain $\mathcal{O}_\theta$ minimizes the error tolerance of S for any value of θ. The error tolerance is given by the formula $\mu(\mathcal{O}_\theta, S) = \max\{d_\theta(e^u_{l_\theta}, e^b_{l_\theta}), d_\theta(e^u_{r_\theta}, e^b_{r_\theta})\}$, where $d_\theta(p, q)$ denotes the distance between parallel lines through p and q with orientation θ.

Let $y_{\min,\theta}$ and $y_{\max,\theta}$ be the minimum and maximum y-coordinates of the points in S when the coordinate system is rotated by an angle θ. The following lemma is a generalization of Lemma 1.

Lemma 3 *An optimal fitting for S by an un-oriented 2-orthogonal polygonal chain with orientation θ is determined by a line ℓ_θ through a point of S separating the points with y-coordinates $y_{\min,\theta}$ and $y_{\max,\theta}$.*

In our approach we adapt the $O(n \log n)$-time algorithm for the oriented 2-fitting case as follows. First, we identify the four quadrants of the coordinate system by its oriented bisectors, i.e., by the four oriented lines with slopes $\tan(\theta{+}45)$, $\tan(\theta{+}135)$, $\tan(\theta{+}225)$, $\tan(\theta + 315)$. Next, we will use the results from Avis et al. [1] about the computation of the so-called un-oriented Θ-maxima.

Definition 2 *A point $p \in S$ is an un-oriented Θ-maximum with respect to S iff there exist rays, C and D, emanating from p with an angle at least Θ between them so that the points of S lie outside the (Θ-angle) cone defined by p, C and D.*

Theorem 4 [1] *Let S be a set of n points in the plane. All un-oriented Θ-maxima points of S for $\Theta \geq \pi/2$ can be computed in $O(n \log n)$ time and $O(n)$ space. The algorithm is optimal for fixed values of Θ.*

The main idea of the algorithm is to rotate the coordinate system by an angle θ, $0 \leq \theta \leq \pi/2$, and update the four staircase structure by inserting or deleting points to each of the four staircases according to the changes of the orientation θ. We can maintain four ordered lists of the current un-oriented $\pi/2$-maxima points of S with respect to the four current quadrants which bisectors have orientations $\theta + 45$, $\theta + 135$, $\theta + 225$, $\theta + 315$. The lists correspond to the points in the four staircases of the staircase structure. Precisely, this staircase structure will be maintained (with insertions and deletions) according to the changes of the value θ.

As θ changes, the four staircases can be modified because a new point of S become $\pi/2$-maxima or some current $\pi/2$-maxima point of S has to be deleted. In order to know the sequence of these events according to the variation of θ, we use Theorem 4 to compute the set of all un-oriented $\pi/2$-maxima points of S in $O(n \log n)$ time together with their respective *orientation intervals*. These orientation intervals are the intervals where each point is un-oriented $\pi/2$-maxima for S. Because we are looking for $\pi/2$-maxima points, we know that a point can be $\pi/2$-maxima for at most 3 (disjoint) orientation intervals. Then, the total number of changes in the staircase structure is linear.

In order to know the sequences of events (insertion and/or deletion of points in the staircase structure as we rotate the coordinate system) we proceed as follows: suppose that we have computed the orientation intervals for each point $p_i \in S$, being $\pi/2$-maxima in some orientation in $[0, 2\pi]$. A point $p \in S$ can have 0, 1, 2 or 3 disjoint orientation intervals. We can sweep the set of these intervals from 0 to 2π with a vertical line, knowing for a particular orientation α which are the set of $\pi/2$-maxima points for that orientation. Analogously, for the orientations $\alpha + 90$, $\alpha + 180$, and $\alpha + 270$. Thus, with the sweeping we can know which is the sequence of the points to be inserted or deleted in the four staircase structure according to the changes of orientation.

Once we have computed the orientation intervals of the $\pi/2$-maxima points of S, the algorithm does a vertical sweep of these intervals (from $\theta = 0$ to $\theta = 90$) with four vertical lines corresponding to the orientations θ, $\theta + 90$, $\theta + 180$, and $\theta + 270$, stoping at each event (endpoint interval), updating the four staircase structures. Since the points are in general position, only a constant number of updates can occur at some event. We compute the optimal solution corresponding to the staircase structure in between two consecutive events θ_1 and θ_2, by computing a line ℓ with slope $\tan(\theta + 90)$. By Lemma 3, for $\theta \in [\theta_1, \theta_2]$, the line ℓ giving the optimal solution for θ has to separate the points with y-coordinates $y_{\min,\theta}$ and $y_{\max,\theta}$. In this way, the optimal solution is determined by two pairs of points, either (i) $(y_{\max,\theta}, e^b_{l_\theta})$ and $(e^u_{r_\theta}, y_{\min,\theta})$ if $y_{\max,\theta}$ is on the left side of $y_{\min,\theta}$, or (ii) $(e^u_{l_\theta}, y_{\min,\theta})$ and $(y_{\max,\theta}, e^u_{r_\theta})$ if $y_{\max,\theta}$ is on the right side of $y_{\min,\theta}$, giving the error tolerance in S_{l_θ} and S_{r_θ}. Now, in order to compute the optimal solution in between two consecutive events, we use the next lemma.

Lemma 5 *As we rotate counterclockwise the coordinate system, the vertical distances giving the error tolerance for the staircases of the first and third quadrants is monotone.*

As a consequence of Lemma 5, the optimal solution for an interval of orientations $\theta \in [\theta_1, \theta_2]$ has to be in the extreme of the interval, so either in θ_1 or in θ_2. Since a point $p \in S$ can become un-oriented $\pi/2$-maximum for at most three disjoint oriented interval, this imply that the number of updates (insertions and deletions) for the entire staircase structure as θ changes from 0 to $\pi/2$ is linear. Any update can be done in $O(\log n)$ time and for a fixed value of θ we can compute in $O(\log n)$ time by binary search the optimal location of the line ℓ with its corresponding error tolerance (maintaining the minimum). We conclude the following result.

Theorem 6 *The un-oriented 2-fitting problem can be solved in $\Theta(n \log n)$ time and $O(n)$ space.*

Notice that the un-oriented 1-fitting problem is equivalent to the problem of computing the width of a point set [7]. The $\Omega(n \log n)$ time lower bound for the un-oriented 1-fitting problem implies the same bound for the un-oriented 2-fitting problem. Nevertheless we have constructed specific reductions to MAX-GAP problem for any given k.

4 Extensions to 3D

We have considered the oriented 2-fitting problem in 3D where a polygonal chain is a configuration of orthogonal planes. For the oriented 2-fitting problem we have algorithms which time complexities are in between $O(n)$ and $O(n^2)$ depending of which information is fixed: (1) fixing the orientation of both the separating plane and the supported planes of the subsets of the bipartition, $O(n)$ time; (2) fixing the orientation of the separating plane, $O(n^2)$ time; and (3) fixing the orientation of the supporting planes, $O(n^2)$ time. See [2] for NP-hard results for the 3D fitting problem.

Acknowledgements. These problems were posed and partially solved during the *Third Spanish Workshop on Geometric Optimization*, July 4-8, 2006, El Rocío, Huelva, Spain.

References

[1] D. Avis, B. Beresford-Smith, L. Devroye, H. Elgindy, E. Guvremont, F. Hurtado, and B. Zhu. Unoriented Θ-maxima in the plane: complexity and algorithms. *SIAM Journal of Computing*, 28 (1999) 278–296.

[2] D. Z. Chen and H. Wang. Approximating points by piecewise linear functions. *18th Fall Workshop on Computational Geometry, 2008, Rensselaer Polytechnic Institute, NY, USA.*

[3] J. M. Díaz-Báñez and J. A. Mesa. Fitting rectilinear polygonal curves to a set of points in the plane. *European Journal of Oper. Research*, 130 (2001), 214–222.

[4] H. Fournier and A. Vigneron. Fitting a step function to a point set. *LNCS 5193*, 2008, 442–453.

[5] M. T. Goodrich. Efficient piecewise-linear function approximation using uniform metric. *Discrete and Computational Geometry*, 14, (1995), 445–462.

[6] S. L. Hakimi and E. F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *Graphical Models and Image Processing*, 53, (1991), 132–136.

[7] D. T. Lee and Y. F. Wu. Geometric complexity of some location problems. *Algorithmica*, 1 (1986) 193–211.

[8] M. A. Lopez and Y. Mayster. Approximating a set of points by a step function. *Journal of Visual Communication and Image Representation*, 17 (2006) 1178-1189.

[9] F. P. Preparata and M. I. Shamos. *Computacional Geometry, an Introduction*, Springer-Verlag, 1988.

[10] D. P. Wang. A new algorithm for fitting a rectilinear x-monotone curve to a set of points in the plane. *Pattern Recognition Letters*, 23 (2006) 329–334.

[11] D. P. Wang, N. F. Huang, H. S. Chao, and R. C. T. Lee. Plane sweep algorithms for polygonal approximation problems with applications. *Lecture Notes in Computer Science*, 762 (1993) 515–522.

Point Distance Problems with Dependent Uncertainties

Yonatan Myers and Leo Joskowicz [*]

Abstract

We address distance problems for planar points whose location is uncertain and possibly coupled. Point coordinate uncertainties are modeled with the Linear Parametric Geometric Uncertainty Model (LPGUM). The model is a general and computationally efficient worst-case first-order linear approximation of geometric uncertainty that allows for dependencies among uncertainties. We formulate closest pair, diameter, width, and bounding box problems in LPGUM, summarize and extend known results for independent point uncertainties, and describe new algorithms for dependent ones.

1 Introduction

Geometric uncertainty is ubiquitous in mechanical CAD/CAM, robotics, computer vision, and many other fields. Sensing, measurement, and manufacturing processes are intrinsically imprecise, and thus require realistic geometric uncertainty models and computationally efficient algorithms. Recent research has focused on modeling and computing with geometric imprecision [1, 3, 7, 9].

We address distance problems for planar points whose location is uncertain and possibly coupled. Point distance problems include closest/farthest point pair, minimum/maximum diameter, and many more. In the nominal case (no uncertainty) these problems are well understood and have optimal solutions. However, when points locations are imprecise, distance problems may be ill-defined and harder to solve depending on the uncertainty model.

Most geometric uncertainty models represent the point location uncertainty as a simple, iso-centric, and independent region of the plane containing the nominal point. Point uncertainty regions can be circles [2, 5], squares [7] or convex polygons [4]. In most cases, point distance problems are efficiently solved with existing algorithms because the regions are simple and the uncertainties are independent of each other [7]. In other cases, the problems are NP-hard [6]. Robust, finite precision, and epsilon geometry techniques are only applicable for very small uncertainties. Sampling-based methods, widely used in engineering, are general but only provide approximate answers and are computationally intensive.

A key drawback of existing geometric uncertainty models is that they cannot model mutually dependent uncertainties, which are very common in practice [10]. For example, part measuring and machining processes rely on reference geometric features such as planes and axes to define other part features and dimensional chains. Thus, feature variations are coupled to variations in the reference and chain features. Assuming independent errors often overestimates the actual error and leads to suboptimal solutions.

We have recently introduced the Linear Parametric Geometric Uncertainty Model (LPGUM) [9]. The model is a general, expressive, and computationally efficient worst-case first-order linear approximation of geometric uncertainty that allows for coupling between uncertainties. Points are defined by common parameters with uncertainty intervals. The uncertainty zones around nominal point locations are defined by uncertainty sensitivity matrices whose entries indicate the point coordinates sensitivity to parameters variations and their dependencies. We have developed efficient algorithms for computing uncertainty zones of points and lines in the plane and for answering relative point and line positioning queries.

In this paper we formulate closest pair, diameter, and bounding box problems in LPGUM, summarize and extend known results for independent uncertainties, and describe new algorithms for dependent ones.

2 LPGUM of a point

The LPGUM of a point in the plane is defined as follows [9]. Let $q = [q_1, q_2, ..., q_k]^{\mathrm{T}}$ be a vector of k parameters over an *uncertainty domain* Δ. Each parameter has an *uncertainty interval* $\Delta_i = [q_i^-, q_i^+]$ from which it can take a value. The parameters' uncertainty domain $\Delta = \Delta_1 \times \Delta_2 \ldots \times \Delta_k$ is the product of the parameters uncertainty intervals. The *nominal parameters vector* $\bar{q} = (\bar{q}_1, ..., \bar{q}_k)$ is the parameters vector values with no uncertainty. WLOG, we assume that the uncertainty intervals are zero-centered symmetric, e.g., $-q_i^- = q_i^+$ and $\bar{q} = 0$ (asymmetric domains are made symmetric by adjusting the nominal parameter value and its interval).

Every point is associated with a $2 \times k$ *uncertainty sensitivity matrix* A_v. The matrix entries are constants that quantify the sensitivity of the point coor-

[*]School of Engineering and Computer Science, The Hebrew University of Jerusalem, ISRAEL. Emails: yoni_m@cs.huji.ac.il, josko@cs.huji.ac.il

dinates to the parameters. Entry $A_v[i,j]$ determines the first-order linear dependence of coordinate i to q_j ($i = 1$ for x, $i = 2$ for y). It is zero when coordinate i is independent of parameter q_j.

The LPGUM of point $v(q)$ is defined by $(\bar{v}, A_v, q, \Delta)$ where $\bar{v}$ is the nominal point location, A_v is the uncertainty sensitivity matrix, q is the parameters vector, and Δ is the parameters uncertainty domain.

The point LPGUM defines the set of possible points locations for instances of parameters vector q:

$$v(q) = \bar{v} + A_v q$$

In the nominal case, $\bar{q} = 0$ and $v(0) = \bar{v}$.

The *uncertainty zone* of v is the region of $\mathbb{R}^2$:

$$\mathcal{Z}(v(q)) = \{v \mid v = \bar{v} + Aq,\ q \in \Delta\}$$

Its boundary is a zonotope (a centrally symmetric polygon) $\mathcal{BZ}(v(q)) = \{z_1, z_2, ..., z_l\}$ with at most $2k$ vertices, $2 \le l \le 2k$. The vertices are point instances of extremal parameters values, q_j^- or q_j^+. The zonotope can be computed in optimal $O(k \log k)$ time [9].

3 Distance between two LPGUM points

Let $u(q)$ and $v(q)$ be two LPGUM points defined by $(\bar{v}, A_v, q, \Delta)$ and $(\bar{u}, A_u, q, \Delta)$, respectively:

$$u(q) = \bar{u} + A_u q$$
$$v(q) = \bar{v} + A_v q$$

Note that both independent and dependent uncertain points can be modeled. Points whose uncertainty is independent are modeled with a parameters vector q consisting of two vectors $q = (q_u, q_v)$ of size k_u and k_v, $k = k_u + k_v$. The $2 \times k$ uncertainty sensitivity matrices are A_u, A_v. The first k_u columns of A_u are constants that reflect the dependence of u coordinates on each parameter of q_u; the remaining k_v columns of A_u are set to zero (no dependence on q_v). Similarly, the first k_u columns of A_v are set to zero; the remaining k_v columns are set to constants that reflect the dependence of v coordinates on each parameter of q_v. Dependent uncertain points are modeled with a joint parameters vector q. The entries of A_u, A_v are set appropriately to reflect the dependencies.

For example, let $\bar{u} = (0, 0)$ and $\bar{v} = (2, 1)$ be two nominal points (Fig. 1). Point u has a ± 1 uncertainty in x and no uncertainty in y; v has no uncertainty in x and ± 1 in y. When their uncertainties are independent, each point is defined by a parameter $q_u, q_v \in [-1, 1]$. Their parameters vector is $q = (q_u, q_v)$ and their uncertainty sensitivity matrices are $A_u = [1\ 0|0\ 0]$ and $A_v = [0\ 0|0\ 1]$. When the uncertainties are dependent, the points are defined by a common parameter $q \in [-1, 1]$ and their sensitivity matrices are $A_u = [1\ 0]^T$ and $A_v = [0\ 1]^T$. Their zonotopes are line segments (u^-, u^+) and (v^-, v^+).

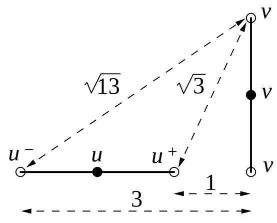


Figure 1: Example of pairwise distances.

The distance between two LPGUM points is the Euclidean distance between the point instances for the **same** parameters vector instance q:

$$dist(u(q), v(q)) = \|u(q) - v(q)\|$$

The minimum (maximum) distance between two LPGUM points is the smallest (largest) distance between point instances for the same parameters vector:

$$\textit{min-dist}(u(q), v(q)) = \min_{q \in \Delta} \|u(q) - v(q)\|$$
$$\textit{max-dist}(u(q), v(q)) = \max_{q \in \Delta} \|u(q) - v(q)\|$$

The minimum and maximum distances can differ for independent and dependent point uncertainties. In Fig. 1, the independent and dependent minimum and maximum distances are 1 (u^+, v^-) and $\sqrt{3}$ (u^+, v^+), and 3 (u^-, v^-) and $\sqrt{13}$ (u^-, v^-) respectively. Note that the dependent scenario distances are tighter than those of the independent one. In fact, it can be shown that the minimum and maximum distances of the independent version of a dependent scenario are its conservative lower and upper bounds.

Minimum and maximum distances between two LPGUM points are bounded quadratic optimization problems solved with quadratic programming.

We now describe an alternative geometric optimal algorithm. Let $w(q)$ be a new LPGUM point $(\bar{w}, A_w, q, \Delta)$ such that $w(q) = u(q) - v(q)$, $\bar{w} = \bar{u} - \bar{v}$, and $A_w = A_u - A_v$. The pairwise maximum distance occurs when $w(q)$ is furtherest from the origin. By convexity, since $w(q)$ is an LPGUM point, its furthest point from the origin is a zonotope vertex. To find it, we compute $\mathcal{BZ}(v(q))$ in $O(k \log k)$, find its furthest vertex from the origin in $O(k)$, and report its parameters vector instance. To find the minimum pairwise distance, we test if the origin is inside the zonotope in $O(k)$. If it is, the minimum distance is zero and its parameters vector is the solution to $\bar{w} + A_w q = 0$. Otherwise we find the closest point of the zonotope to the origin in $O(k)$ and report its parameters vector.

4 Distance problems for n LPGUM points

Let $\mathcal{V}(q) = \{v_1(q), ..., v_n(q)\}$ be a set of n LPGUM points, each defined by $(\bar{v}_i, A_{v_i}, q, \Delta)$.

Problem	Model	Smallest	Largest
Closest Pair	independent	$O(nk\log nk)$ [7]*	NP-hard [6]
	dependent	$O\left(n^2k\log k\right)$	
Diameter	independent	$O(nk\log nk)$ [7]*	$O(nk\log nk)$ [7]*
	dependent		$O\left(n^2k\log k\right)$
Width	independent	$O(nk\log nk)$ [7]*	NP-hard [7]
	dependent		
Axis-Aligned Bounding Box	independent	$O(nk)$ [7]*	$O(nk)$ [7]*
	dependent perimeter	$O(n^4LP(k))$	$O(n^4LP(k))$
	dependent area	$O(n^4QP(k))$	$O(n^4QP(k))$

Table 1: Time complexities of distance problems for n LPGUM points with independent and dependent uncertainties, where k is the maximum number of parameters of a point (independent) or the total number of parameters of all points (dependent) and $LP(k)$, $QP(k)$ are the k-variable LP and QP problem complexities. * Extention of [7]

For a given parameters vector instance q, $\mathcal{V}(q)$ is the set of points obtained by substituting q in each point LPGUM. The minimum and maximum distances of $\mathcal{V}(q)$ are the smallest and largest pairwise distance between the corresponding point distances:

$$\textit{min-dist}(\mathcal{V}(q)) = \min_{i,j} \; \textit{dist}\left(v_i(q), v_j(q)\right)$$
$$\textit{max-dist}(\mathcal{V}(q)) = \max_{i,j} \; \textit{dist}\left(v_i(q), v_j(q)\right)$$

When parameters vector q is unspecified, $\mathcal{V}(q)$ is a set of uncertainty zones $\mathcal{Z}(v_i(q))$. Point distance problems involve finding the parameters vector instance q and points $v_i(q), v_j(q)$ that maximize or minimize the pairwise point instances distances. This raises a family of distance problems, which we formulate next.

Problem 1 Smallest possible distance
The smallest distance of all point pairs instances

$$\textit{smallest-dist}(\mathcal{V}(q)) = \min_{q\in\Delta} \textit{min-dist}(\mathcal{V}(q))$$

Problem 2 Largest possible distance
The largest distance of all point pairs instances

$$\textit{largest-dist}(\mathcal{V}(q)) = \max_{q\in\Delta} \textit{max-dist}(\mathcal{V}(q))$$

Problem 3 Largest smallest distance
The largest distance of closest point pairs instances

$$\textit{largest-smallest-dist}(\mathcal{V}(q)) = \max_{q\in\Delta} \textit{min-dist}(\mathcal{V}(q))$$

Problem 4 Smallest largest distance
The smallest distance of furthest point pairs instances

$$\textit{smallest-largest-dist}(\mathcal{V}(q)) = \min_{q\in\Delta} \textit{max-dist}(\mathcal{V}(q))$$

These four distances provide useful information about the uncertain points set. The smallest and largest possible distances indicate how close and how far two point instances can ever get. They are the lower and upper bounds on pairwise point distances and the diameters of the minimum enclosed and maximum enclosing circles. The corresponding point pairs are the closest and furthest point pairs. The largest smallest and smallest largest distances indicate how far the closest points and how close the furthest points can ever get. They are the upper and lower bounds on the smallest and largest pairwise point distances and the maximum diameter of the minimum enclosed circle and minimum diameter of the maximum enclosing circle. Note that the parameters vector instances that achieve these distances might all be different.

Closely related problems concern the width and the Axis Aligned Bounding Box (AABB). The smallest (largest) width of $\mathcal{V}(q)$ is the smallest (largest) distance between any pair of parallel lines each supported by points $v_i(q), v_j(q)$ that contain all points instances (the definition of *p-dist* is omitted for lack of space).

Problem 5–6 Smallest and largest width
The smallest/largest parallel strip width

$$\textit{min-width}(\mathcal{V}(q)) = \min_{i,j} \; \textit{p-dist}\left(v_i(q), v_j(q)\right)$$
$$\textit{max-width}(\mathcal{V}(q)) = \max_{i,j} \; \textit{p-dist}\left(v_i(q), v_j(q)\right)$$

The smallest (largest) AABB is the smallest (largest) perimeter/area axis-aligned rectangle that contains all points instances $v_i(q)$. Let $v_{ix}(q), v_{iy}(q)$ be the x, y coordinates of $v_i(q)$. The AABB width and height are:

$$\textit{width}(\mathcal{V}(q)) = \max_{i,j} \left(v_{ix}(q) - v_{jx}(q)\right)$$
$$\textit{height}(\mathcal{V}(q)) = \max_{i,j} \left(v_{iy}(q) - v_{jy}(q)\right)$$

Problem 7–10 Smallest and largest AABB
The smallest/largest perimeter/area AABB

$$\textit{smallest-pb}(\mathcal{V}(q)) = \min_{q\in\Delta}(\textit{width}(\mathcal{V}(q)) + \textit{height}(\mathcal{V}(q)))$$
$$\textit{largest-pb}(\mathcal{V}(q)) = \max_{q\in\Delta}(\textit{width}(\mathcal{V}(q)) + \textit{height}(\mathcal{V}(q)))$$

$$smallest\text{-}ab(\mathcal{V}(q)) = \min_{q\in\Delta}(width(\mathcal{V}(q)) \cdot height(\mathcal{V}(q)))$$
$$largest\text{-}ab(\mathcal{V}(q)) = \max_{q\in\Delta}(width(\mathcal{V}(q)) \cdot height(\mathcal{V}(q)))$$

We distinguish between independent and dependent point uncertainty models. Points with independent uncertainties are modeled with n independent parameters vectors q_i of size k_i such that $q = (q_1, .., q_n)$, $\sum k_i = k$. Their uncertainty regions are bounded by zonotopes $\mathcal{BZ}(v_i(q)) = \{z_{i1}, \dots z_{il}\}$, $1 \le i \le n$, $2 \le l_i \le 2k_i$. Distance problems reduce to finding distances between possibly overlapping convex polygons whose total complexity is $O(nk)$. Previous works address these problems for discs and squares [7]. In many cases the results extend directly to convex polygons. Distance problems for points with dependent uncertainties have not been studied and are often harder to solve.

5 Solutions to LPGUM point distance problems

Table 1 summarizes known and new results [7].

The smallest/largest closest pair distance (Problems 1 and 2) are solved in $O(n^2k \log k)$ by computing the smallest/largest distance between all point pairs $min\text{-}dist(v_i(q), v_j(q))$ and $max\text{-}dist(v_i(q), v_j(q))$. For independent point uncertainties the problems reduce to finding the smallest/largest pairwise distance of n convex polygons, which can be solved in $O(nk \log nk)$ with a direct extension of the algorithms in [7].

The largest closest pair distance (Problem 3) is NP-hard for independent points whose uncertainties are squares [6] and is thus also NP-hard for LPGUM points. The smallest largest distance (Problem 4) for independent point uncertainties is solved in $O(nk \log nk)$ for $k = \max\{k_i\}$ [8]. The problem is open for dependent point uncertainties.

The largest width (Problem 6) is NP-hard for arbitrarily oriented line segments [7], so it is also NP-hard for LPGUM points. The smallest width (Problem 5) for independent point uncertainties is solved in in $O(nk \log nk)$ with convex hull and rotating calipers techniques. The problem is open for dependent point uncertainties.

The smallest/largest AABB perimeter/area (Problems 7–10) for independent point uncertainties are solved in $O(nk)$ by independently computing the maximum height and maximum width [7]. For dependent point uncertainties, the smallest/largest AABB perimeter/area optimization problems are (replace min by max for the largest AABB):

$$\min_{q\in\Delta} \max_{i,j,k,l} ((v_{ix}(q) - v_{jx}(q)) + (v_{ky}(q) - v_{ly}(q)))$$
$$\min_{q\in\Delta} \max_{i,j,k,l} ((v_{ix}(q) - v_{jx}(q)) \cdot (v_{ky}(q) - v_{ly}(q)))$$

A direct solution is to solve k-variable Linear Programming (LP) and Quadratic Programming (QP) problems for each point quadruple in $O(n^4LP(k)$ and $O(n^4QP(k))$, where $LP(k)$ and $O(n^4QP(k))$ are the times required to solve the LP and QP problems.

It remains an open question if these problems can be solved more efficiently with a geometric approach.

6 Conclusion

Dependent point uncertainties are common in practice and have not been systematically studied. We formulate closest pair, diameter, width, and bounding box problems in the Linear Parametric Geometric Uncertainty Model (LPGUM), summarize and extend known bounds and algorithms for independent point uncertainties, and describe new ones for dependent point uncertainties. In most cases, the added complexity is sub-quadratic in the number of parameters and points, with higher complexities for dependent point uncertainties. We are currently developing algorithms to compute the width, convex hull, and Voronoi diagram of LPGUM points.

References

[1] M. Abellanas, F. Hurtado and P. Ramos. Structural tolerance and Delaunay triangulation. *Information Processing Letters* 71:221–227, 1999.

[2] S. Akella and M. Mason. Orienting toleranced polygonal parts. *Int. J. Robotics Res* 19(12):1147–70, 2000.

[3] I. Averbakh and S. Bereg. Facility location problems with uncertainty on the plane. *Disc. Opt.*2:3-34, 2005.

[4] R. C. Brost and R. R. Peters. Automatic design of 3D fixtures and assembly pallets. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp 495–502, USA , 1996.

[5] J. Chen, K. Goldberg, M. Overmars, D. Halperin, K.-F. Böhringer, and Y. Zhuang. Computing tolerance parameters for fixturing and feeding. *The Assembly Automation Journal* 22:163–172, 2002.

[6] J. Fiala, J. Kratochvíl, and A. Proskurowski. Systems of distant representatives. *Discrete Appl. Math.* 145(2):306–316, 2005.

[7] M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Algorithms and Data Structures*, *Lecture Notes in Computer Science* 4619, pp 446–457. Springer, 2007.

[8] N. Meggido. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Computing* 12(4):759–776, 1983.

[9] Y. Myers and L. Joskowicz. The linear parametric geometric uncertainty model: points, lines and their relative positioning. Proc. *24th European Workshop on Computational Geometry* pp 137– 140, France, 2008.

[10] Y. Ostrovsky-Berman and L. Joskowicz. Tolerance envelopes of planar mechanical parts with parametric tolerances. *Computer-Aided Design* 37(5):531-44, 2005

An Improved Algorithm for Inserting a Highway in a City Metric Based on Minimization of Quasiconvex Functions

Matias Korman[*] Takeshi Tokuyama[*]

Abstract

We introduce an improved algorithm to locate a segment highway such that the maximum city distance between a given set of n points is minimized (where the city distance is measured with speed $v > 1$ on a highway and 1 in the underlying metric elsewhere). We consider that such highway is built in a complex transportation system with H other highways and obstacles. The algorithm runs in $O(n^3H^3(\log^2 n + \log H))$ time using $O(nH)$ space improving the previous $O(n^4H^4)$ time and space bound.

1 Introduction

We consider a city that has some kind of transportation facility (i.e., bus, train, ...) that speeds up the travel time of its passengers. Imagine that we travel in such a city starting from p towards a destination q: we normally should not go along the geometric shortest path, which is the straight line segment pq; instead, we may take a detour to ride a train, and we must also avoid obstacles like buildings. We model the transportation system by a set of *highways* and *obstacles*. Under that environment we would like to, given a set of points $\mathcal{S}$, a set of polygonal obstacles $\mathcal{O}$, and a set of existing highways $\mathcal{H}$, locate a new highway to reduce the diameter of $\mathcal{S}$ (i.e: the maximum distance among all possible pairs of points in $\mathcal{S}$).

Cardinal and Langerman considered the problem of locating a line highway (of infinite length) to reduce the diameter of a given set of n points without any prelocated highways/obstacles if $v = \infty$ in [5] . Their algorithm has quadratic time complexity and was further improved by Ahn *et al.* in [1]: $\Theta(n)$ time to locate an axis aligned highway (of any speed) and $\Theta(n \log n)$ if the highway can be arbitrarily oriented. Further research by Cardinal *et al.* [4] gave a $\Theta(n \log n)$ algorithm to locate an axis aligned segment highway using LP optimization techniques.

The above algorithms can only be used if no highways have been built. Thus, by applying any of those methods we obtain a transportation system with only one highway. Several optimization problems of inserting a new highway in a city metric or generalized city metric were considered in [7] and [8]. In particular, an $O(n^4H^4)$ time and space algorthm was proposed in [8] to minimize the city diameter by optimally locating an axis-alighed highway of unit length in an environment with H prelocated highways.

In this paper, we focus on the case that the underlying metric is L_1, all highways axis parallel of equal speed and all obstacles are nonoverlapping open polygons. The induced metric is called the *generalized city metric* [3] (*city metric* if there is no obstacle). We note that our algorithm could work under other metrics, allow overlapping obstacles and a constant number of speeds and orientations for the highways without much modification.

1.1 Notations

A *highway* is a facility supporting faster movement along it, which is represented by an axis aligned segment on the plane. Given an underlying metric d, each highway h is associated with its speed $v(h) > 1$. Under the *time metric*, a traveler moves at speed $v(h)$ when moving along highway h, and at the unit speed otherwise. We consider a highway only accessible through its endpoints[1]. An *obstacle* is a region that a traveler is not allowed to cross, and represented by a simple polygon. We note that a different kind of highway that allows access through any point (called *freeway*) has also been considered. Our algorithm can be generalized to work in a hybrid environment with both kinds of highways, but for simplicity in the exposition we assume that no freeway exists. In Figure 1 we can see a sample problem with four points, four highways and an obstacle (and the diameter as a dashed polygonal chain). In Figure 2, a new highway h has been located and the diameter has been reduced.

A feasible path is a polygonal path avoiding all obstacles in $\mathcal{O}$. The travel time of any feasible path π between two points is the sum of travel time of edges of π obtained by dividing the length of the edge in the underlying metric by the travel speed associated to each edge. We call a feasible path π connecting s and t a *shortest (travel time) path* if π minimizes the travel time between s and t. Let $d_{\mathcal{H},\mathcal{O}}(s,t)$ be the

[*]Graduate School of Information Sciences, Tohoku University, Sendai, Japan, {mati,tokuyama}@dais.is.tohoku.ac.jp

[1]This highway type has been named *walkway* or *turnpike* [7, 4] in the literature.

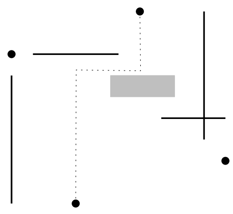


Figure 1: Example of generalized city metric

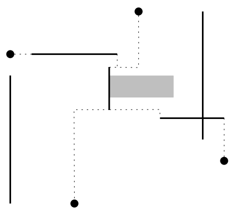


Figure 2: Updated diameter once h is inserted

travel time of a shortest path between s and t induced by $\mathcal{H}$ and $\mathcal{O}$.

Without loss of generality, we consider that the new highway to build h is vertical of unit length (since we can rotate and scale the problem to transform the general problem into this one). The insertion of such highway h is parameterized by the location of its bottom endpoint β. We denote the inserted highway h_β and $\mathcal{H}_\beta$ for the updated set of highways (i.e.: $\mathcal{H}_\beta = \mathcal{H}\bigcup\{h_\beta\}$). Regarding $\beta = (x, y)$ as a function of its two coordinates, we define $F_{s,t}(x, y) = d_{\mathcal{H}_{(x,y)},\mathcal{O}}(s, t)$ for any pair of points $s, t \in \mathcal{S}$. The diameter D is the minimum value of the upper envelope of the set $\{F_{s,t}|s, t \in \mathcal{S}\}$, which is a bivariate function. We will bound the running time of our algorithms by n and H, the number of points in $\mathcal{S}$ and the complexity of the transportation network (highways and obstacles), respectively.

2 Framework using quasiconvex optimization

Although time metric diameter minimization problems are not LP problems in general, Cardinal et.al. [4] proposed a novel quasi-convex optimization method to solve a diamater minimization problem that works under the city metric. Their method is based the randomized parametric searching methodology of Chan [6] . A function f is called *quasiconvex* if its lower level sets are convex (i.e.: the set $\{x|f(x) < t\}$ is convex for any value of t). It was shown in [4] that functions $F_{s,t}$ are quasconvex if $\mathcal{O} = \mathcal{H} = \emptyset$. The main tool is the following theorem obtained from a more general result given in[4](Theorem 5):

Theorem 1 *For any given $\mathcal{S}$, the point $p \in \mathbb{R}^2$ that minimizes $F(p) = \max_i\{F_{s,t}(p)|s, t \in \mathcal{S}\}$ can be computed in $O(P(n))$ time using $O(n)$ space, provided that we can compute $F(p)$ for any $p \in \mathbb{R}^2$ in $O(P(n))$ time and $P(n)/n^\epsilon$ is monotone increasing in n for a suitable constant $\epsilon > 0$.*

Our overall plan is to generalize this idea to our problem. Unfortunately, such method is not directly applicable, since quasiconvexity does not hold if one or more prelocated highway or obstacle exists (see figure 3). Nevertheless, we can utilize Theorem 1 by partitioning the plane into regions such that quasiconvexity of $F_{s,t}$ restricted to each region is certified. Having local quasiconvexity, we will apply Theorem 1 for each region independently. By iterating among all possible regions, the global optimal is obtained.

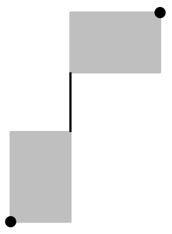


Figure 3: Non-quasiconvexity of $F_{s,t}$ functions: location is optimal if the new highway is inside either of the grey rectangles.

For the purpose, Theorem 1 must be adapted to work in a restricted region:

Lemma 2 *For any given $\mathcal{S}$, $\mathcal{H}$ and a convex region c, the point $p \in \mathbb{R}^2$ that minimizes $F(p) = \max_i\{F_{s,t}(p)|s, t \in \mathcal{S}\}$ can be computed in $O(P(n, H))$ time using $O(n+H)$ space, provided that we can compute $F(p)$ for any $p \in \mathbb{R}^2$ in $O(P(n, H))$ time and $P(n, H)/n^\epsilon$ is monotone increasing in n for a suitable constant $\epsilon > 0$.*

We apply the *randomized parametric search* paradigm of Chan [6]. We omit details of the proof in this version, since it is a direct consequence of Chan's method.

We say that a plane subdivision $\mathcal{R}$ is *admissible* if for any cell $c \in \mathcal{R}$ and $s, t \in \mathcal{S}$, function $F_{s,t}$ is quasiconvex in c. In the following, we construct of an admisible subdivision $\mathcal{A}_\mathcal{H}$ of low complexity. We can then use an efficient algorithm (Theorem 3 given below) to compute the diameter to design our algorithm for finding the optimal location of the highway.

Theorem 3 **([3])** *The city diameter of a set of points $\mathcal{S}$ of n points can be computed in $O(nH(\log^2 n + \log H))$ time using $O(n + H)$ space.*

The time complexity of the algorithm in the above theorem becomes $P(n, H)$. Lemma 2 implies that for each cell c in $\mathcal{A}_\mathcal{H}$, we can compute the optimal location of (the lowest point of) newly inserted vertical highway in $O(P(n,H))$ time, and we simply apply the method for every cell of c. Thus, if the complexity of $\mathcal{A}_\mathcal{H}$ is $Q(n,H)$, we can find the optimal location of the inserted highway in $O(P(n,H)Q(n,H)) = O(Q(n,H)nH(\log^2 n + \log H))$ time. Thus, it suffices to construct an admissible $\mathcal{A}_\mathcal{H}$ and analyze $Q(n,H)$.

3 Construction of an admissible subdivision

The shortest path map is the subdivision of a plane into regions of points with the same combinatorial strucure of shortest paths towards a destination (or a source) introduced by Mitchell in [9]. In our case, the shortest path map with the source point $s \in \mathbb{R}^2$ ($SPM(s,\mathcal{H},\mathcal{O})$ for short) is defined as the subdivision of the plane into cells such that all the points in one cell have combinatorially equivalent paths to s. We are not only interested in combinatorial equivalence of paths, but also the exact distance functions: for any $s \in \mathcal{S}$ we define $f_s(x,y) = d_{\mathcal{H},\mathcal{O}}(s,(x,y))$ representing the distance between a fixed point s and $p=(x,y)$. The following Theorem is given by Bae *et al.*:

Theorem 4 ([2]) *$SPM(s,\mathcal{H},\mathcal{O})$ has $\Theta(H)$ complexity, and can be constructed in $\Theta(H\log H)$ time. Moreover, in each cell c of $SPM(s,\mathcal{H},\mathcal{O})$), the function f_s is linear and its coefficients are $\{\pm 1, \pm 1/v\}$ (i.e.: $f_s(x,y)|_c = k_1x + k_2y + k_3$ where $k_1, k_2 \in \{\pm 1, \pm 1/v\}$ and $k_3 > 0$).*

We note that the Theorem stated in [2] considered a transportation network with freeways and obstacles (no turnpikes), but it was adapted for the generalized city metric in [3]. The computation of $SPM(s,\mathcal{H},\mathcal{O})$ simulates the wavefront propagation from s: the *wavefront* is defined as $W(\delta) = \{p \in \mathbb{R}^2 \mid d_{\mathcal{H},\mathcal{O}}(p,s) = \delta\}$ for any positive δ. In particular, under the L_1 metric, the wavefront $W(\delta)$ is a set of line segments, called *wavelets*. Subdivision $SPM(s,\mathcal{H},\mathcal{O})$ is computed by tracking the topological and geometric changes of $W(\delta)$, since its vertices trace the edges of the $SPM(s,\mathcal{H},\mathcal{O})$.

We consider the translation of a region R : let $e_\lambda = (0,\lambda)$, we define $R + e_\lambda = \{(x,y) \in \mathbb{R}^2 | (x,y) + e_\lambda \in R\}$. We also define $\mathcal{R} + e_\lambda = \{R_1 + e_\lambda, \dots R_k + e_\lambda\}$ for a plane subdivision $\mathcal{R} = \{R_1, \dots R_k\}$. That is: subdivision $\mathcal{R} + e_\lambda$ is the result of vertically shifting down λ units subdivision $\mathcal{R}$. We also define the translation of a bivariate function $f(x,y)$ by one unit: let $f^+(x,y) = f(x,y+1)$.

We define $\mathcal{A}_\mathcal{H}$ as the overlay the regions $SPM(s,\mathcal{H},\mathcal{O})$, $SPM(s,\mathcal{H},\mathcal{O}) + e_1$, $SPM(s,\mathcal{H},\mathcal{O}) + e_{(1/2-1/2v)}$ and $SPM(s,\mathcal{H},\mathcal{O}) + e_{(1/2v-1/2)}$ for every $s \in \mathcal{S}$. By definition, two points are in the same cell of $\mathcal{A}_\mathcal{H}$ if they are in the same cell of the $4n$ subdivisions. $\mathcal{A}_\mathcal{H}$ can be constructed by first computing $SPM(s,\mathcal{H},\mathcal{O})$ for each s and then their union (together with its vertically shifted copies) by applying the plane sweep method. Total cost of the algorithm is $O(Q(n,H)\log nH)$ computational time. Since we are overlaying $4n$ different regions each of which has size $O(H)$, we have $Q(n,H) \in O(n^2H^2)$.

Before proving admissibility of $\mathcal{A}_\mathcal{H}$, we need some observations: for any fixed $s,t \in \mathcal{S}$, let $\pi_1 = d_\mathcal{H}(s,t)$, $\pi_2(\beta) = f_s(\beta) + 1/v + f_t^+(\beta)$ and $\pi_3(\beta) = f_s^+(\beta) + 1/v + f_t(\beta)$. We have:

Lemma 5 *For any $s,t \in \mathcal{S}$, $\beta \in \mathbb{R}^2$, we have $F_{s,t}(\beta) = \min\{\pi_1, \pi_2(\beta), \pi_3(\beta)\}$. Moreover, if $\pi_2(\beta) < \pi_1$, we have $f_s(\beta) + 1/v < f_s^+(\beta)$ (analogously, if $\pi_3(\beta) < \pi_1$ then $f_s^+(\beta) + 1/v < f_s(\beta)$).*

Proof. The shortest path between s and t in the updated configuration either does not use the new highway (and then $F_{s,t} = d_\mathcal{H}(s,t)$) or uses the highway *upward* ($F_{s,t} = f_s(\beta) + 1/v + f_t^+(\beta)$) or the highway is used *downward* ($F_{s,t} = f_s^+(\beta) + 1/v + f_t(\beta)$). The second claim is proved by the fact that $\pi_1 = d_\mathcal{H}(s,t) \le f_s^+(\beta) + f_t^+(\beta)$ and thus $\pi_2(\beta) < \pi_1 \implies f_s(\beta) + 1/v + f_t^+(\beta) < f_s^+(\beta) + f_t^+(\beta) \iff f_s(\beta) + 1/v < f_s^+(\beta)$ (analogously for π_3). ☐

We say that two points $\beta_2, \beta_3 \in \mathbb{R}^2$ are *opposite* if there exist $s,t \in \mathcal{S}$ such that the shortest path of s and t uses the new highway upward in $\mathcal{H}_{\beta_2}$ and downward in $\mathcal{H}_{\beta_3}$. We have the following:

Lemma 6 *There cannot be a pair of opposite points in the same cell of $\mathcal{A}_\mathcal{H}$.*

Proof. Proof of the lemma is by contradiction: assume that there exists $s,t \in \mathcal{S}$, cell $c \in \mathcal{A}_\mathcal{H}$, and two points $\beta_2, \beta_3 \in c$ that satisfy $F_{s,t}(\beta_2) = \pi_2(\beta_2)$ and $F_{s,t}(\beta_3) = \pi_3(\beta_3)$. Let ℓ be the bisector of functions f_s and f_s^+: by Theorem 4 and definition of $\mathcal{A}_\mathcal{H}$, both functions are linear in c, (therefore ℓ must also be a line). Also, by lemma 5, points β_2 and β_3 must be on different sides of ℓ. Without loss of generality, we assume ℓ is parallel to line $x + y = 0$ and that β_3 is in the upper halfplane of the bisector, see Figure 4.

Let β_1 be any point in ℓ and consider the creation of $SPM(s,\mathcal{H},\mathcal{O})$ in the neighborhood of β_1: since the distance between β_1 and $\beta_1 + (0,1)$ is one unit and both points are equidistant to s, their respective wavelets are bound to collide at $\delta = f_s(\beta_1) + 1/2$. The collision might actually not occur if a third wavelet collides with either wavelet earlier. In either case, we can certify that the wavelet that contains β_1 will collide with another wavelet in at most half a unit afterwards. That collision of wavelets will generate an edge $\nu \in SPM(s,\mathcal{H},\mathcal{O})$, and thus subsequently edge $\nu + (0, -1/2 + 1/2v)$ (or a refinement of it) will also be present in $\mathcal{A}_\mathcal{H}$. That is, all points $p \in c$ that satisfy $f_s > f_s^+$ (and in particular β_3) have vertical

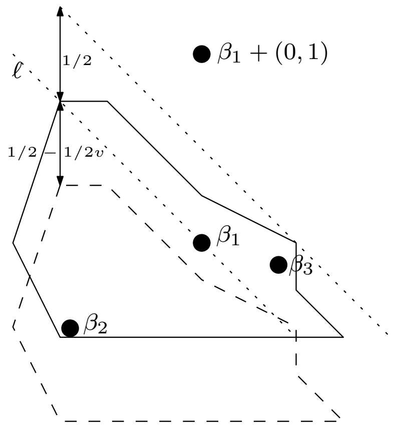


Figure 4: Proof of lemma 7: There cannot be a cell in $SPM(s, \mathcal{H}, \mathcal{O})$ with a point more than 1/2 above the bisector $\ell : f_s - f_s^+ = 0$.

distance to the bisector smaller or equal than $1/2v$ to ℓ.

Consider now the value of $f_s(\beta_3) - f_s^+(\beta_3)$: since both functions are linear of coefficients in $\{\pm 1, \pm 1/v\}$ and its distance to ℓ is less than $1/2v$, we have $|f_s(\beta_3) - f_s^+(\beta_3)| < 2/2v < 1/v$, but that contradicts second claim of lemma 5 for β_3. □

Lemma 7 *Subdivision $\mathcal{A}_\mathcal{H}$ is admissible.*

Proof. Combining the first claim of lemma 5 with lemma 6 we obtain that $F_{s,t}$ is either $F_{s,t}(\beta) = \min\{\pi_1, \pi_2(\beta)\}$ or $F_{s,t}(\beta) = \min\{\pi_1, \pi_3(\beta)\}$ for each $c \in \mathcal{A}_\mathcal{H}$. By Theorem 4, both π_2 and π_3 are linear (and π_1 is a constant), thus the lemma follows. □

We combine all results to obtain the following:

Theorem 8 *The optimal location of a new turnpike can be computed in $O(n^3H^3(\log^2 n + \log H))$ time using $O(nH)$ space.*

Proof. The algorithm computes and makes a trapezoidal subdivision of $\mathcal{A}_\mathcal{H}$. We can then use lemma 2 in each generated cell $c \in \mathcal{A}_\mathcal{H}$ (where the set of lemma 2 to find its optimal. For each cell we spend $O(nH(\log^2 n + \log H))$ time. Space needed can be reduced to $O(nH)$ since an explicit construction of $\mathcal{A}_\mathcal{H}$ is not needed.

□

4 Concluding Remarks

The study of the complexity $\mathcal{A}_\mathcal{H}$ is an interesting problem: we used a naive $O(n^2H^2)$ bound to obtain the upper bound of our algorithms. Under the Euclidean metric, there are examples in which the bound is tight, but the same examples lead to an $\Omega(n^2 + H^2)$ lower bound in the Manhattan metric.

Also, it would be interesting to find a subdivision that preserves quasiconvexity of lower complexity than $\mathcal{A}_\mathcal{H}$. Although only quasiconvexity is needed to use Theorem 1, we proved a much stronger property of $F_{s,t}$ functions, thus such relaxation seems possible.

Acknowledgments

The authors would like to thank J. S. B. Mitchell for helpful discussions on the complexity of $\mathcal{A}_\mathcal{H}$.

References

[1] H.-K. Ahn, H. Alt, T. Asano, S. W. Bae, P. Brass, O. Cheong, C. Knauer, H.-S. Na, C.-S. Shin, and A. Wolff. Constructing optimal highways. In J. Gudmundsson and B. Jay, editors, *Thirteenth Computing: The Australasian Theory Symposium (CATS2007)*, volume 65 of *CRPIT*, pages 7–14, Ballarat, Australia, 2007. ACS.

[2] S. W. Bae, J.-H. Kim, and K.-Y. Chwa. Optimal construction of the city Voronoi diagram. In *Proc. 17th Annu. Internat. Sympos. Algorithms Comput.*, volume 4288 of *LNCS*, pages 183–192, 2006.

[3] S. W. Bae, M. Korman, and T. Tokuyama. All farthest neighbors in the presence of highways and obstacles. In *Proc. 3rd Intl. Workshop on Algorithms and Computation (WALCOM), to appear.*, 2009.

[4] J. Cardinal, S. Collette, F. Hurtado, S. Langerman, and B. Palop. Optimal location of transportation devices. *Comput. Geom. Theory Appl.*, 41(3):219–229, 2008.

[5] J. Cardinal and S. Langerman. Min-max-min geometric facility location problems. In *Proc. 19th Euro. Workshop on Comput. Geom.*, 2006.

[6] T. M. Chan. An optimal randomized algorithm for maximum tukey depth. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 430–436, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[7] M. Korman and T. Tokuyama. Optimal insertion of a segment highway in a city metric. In *Proc. 14th Annu. Computing and Combinatorics (COCOON)*, 2008.

[8] M. Korman and T. Tokuyama. Optimal insertion of a segment highway in a city metric. In *Proc. 21st Euro. Workshop on Comput. Geom.*, 2008.

[9] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

Fréchet Distance in Weighted Regions [*]

Yam Ki Cheung[†] and Ovidiu Daescu[†]

Abstract

We discuss the Fréchet distance problem for two polygonal curves in weighted planar subdivisions, where the distance between two points is the weighted length of the line segment joining the points. We give an algorithm that computes an approximate distance that is within an $(1+\epsilon)$-factor from the Fréchet distance between the curves.

1 Introduction

Measuring similarity between curves is a fundamental problem that appears in various applications, including computer graphics and computer vision, pattern recognition, robotics, and structural biology. One common choice for measuring the similarity between curves is the *Fréchet distance*, introduced by Fréchet in 1906 [9].

The Fréchet distance δ_F for two parametric curves P,Q: $[0,1] \to \mathcal{R}^d$ is defined as

$$\delta_F(P,Q) = \inf_{\alpha,\beta:[0,1]\to[0,1]} \sup_{r\in[0,1]} S(P(\alpha(r)), Q(\beta(r))),$$

where α and β range over all continuous non-decreasing functions with $\alpha(0) = \beta(0) = 0$, $\alpha(1) = \beta(1) = 1$, $P(\alpha(r))$ and $Q(\beta(r))$ are two points on P and Q respectively, S is a distance metric between points, and $d > 0$ is the dimension of the problem. The Fréchet distance is described intuitively by a man walking a dog on a leash. The man follows a curve (path), and the dog follows another path. Both can control their speed but backtracking is not allowed. The Fréchet distance between the curves is the length of the shortest leash that is sufficient for the man and the dog to walk their paths from start to end.

We use $P(s)Q(t)$ to denote the leash (line segment, also called link) with end points $P(s)$ and $P(t)$, where $s, t \in [0,1]$. We call the functions α, β reparameterizations. We say that a pair of reparameterizations α and β defines a *matching* between P and Q or α and β define a *monotone walk* from leash $P(0)Q(0)$ to leash $P(1)Q(1)$.

[*]This work was supported in part by NSF award CCF-0635013.

[†]Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA, Email: {ykcheung,daescu}@utdallas.edu.

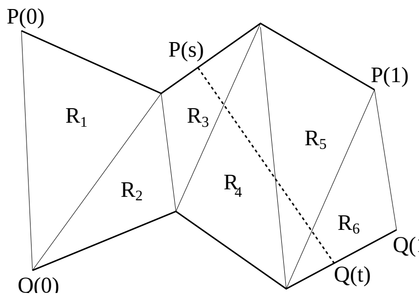


Figure 1: An example of the Fréchet distance problem in weighted regions.

A weighted subdivision is a partition of the plane into polygonal regions, with each region $R_i \in R$ having associated a positive integer weight w_i. Given a line segment st in R, we define the weighted length $S(st)$ of st, also called the weighted distance between s and t, as $S(st) = \sum_{st\cap R_j\neq\emptyset} w_j * d_j(st)$, where $d_j(st)$ is the Euclidean length of st within region R_j.

In this paper, we study the *Fréchet distance problem in weighted regions* in the plane: Given a weighted subdivision $R = \{R_1, R_2, \dots, R_{n'}\}$ with a total of n vertices and two parameterized polygonal chains P and Q in R, find the (weighted) Fréchet distance between P and Q, where the length of the leash is defined as the weighted distance between the end points of the leash. That is, $S(P(s), Q(t)) = \sum_{i=1}^{n'} w_i * d_i(P(s)Q(t))$, where $d_i(P(s)Q(t))$ is the Euclidean length of $P(s)Q(t)$ within region R_i. We assume that no edges of the polygonal curves cross boundaries of R. That is, if a curve crosses a boundary of R, the corresponding intersection between the curve and R is a vertex of the curve. Without loss of generality, we assume R is triangulated and P and Q lie on boundaries of weighted regions (see Fig. 1 as an illustration).

Results. We give an algorithm that computes an approximate distance that is within an $(1+\epsilon)$-factor from the Fréchet distance between the curves. Our algorithm discretizes the problem by placing Steiner points along the edges of R. The running time of the algorithm is $O(N^4 \log N)$, where $N = (1/\epsilon)\log(1/\epsilon)n\log n$ is the number of Steiner points used.

Previous Work. Previous work on the Fréchet distance assumes an unweighted environment and can be divided into two categories, depending on the distance metric used.

In the first category, the distance between two points is the Euclidean distance. In other words, the leash is always a line segment. Fréchet distances in this category are also referred to as non-geodesic Fréchet distances. Alt and Godau [2] give a fundamental study on the computational properties of the Fréchet distance. They present an algorithm that computes the exact Fréchet distance between two polygonal curves in $O(pq \log pq)$ time, where p and q are the number of vertices of P and Q, respectively. Either and Mannila [8] show how to find the discrete Fréchet distance between two polygonal curves. Rote [11] gives algorithms for finding the Fréchet distance between piecewise smooth curves. Buchin et al. [4] study the Fréchet distance between polygons. Wang [13] presents the first exact, polynomial-time algorithm to compute a partial matching between two polygonal curves via Fréchet distance.

In the second category, the distance between two points is the geodesic distance. Fréchet distances in this category are also referred to as geodesic Fréchet distances. Maheshwari and Yi [10] study the case in which the curves lie on a convex polyhedron. Cook and Wenk [6] show how to compute the Fréchet distance between two curves when the leash is constrained inside a polygon or when polygonal obstacles are present between curves. Chambers et al. [5] give a polynomial-time algorithm to compute the *homotopic* Fréchet distance between two polygonal curves in the presence of obstacles. The homotopic Fréchet distance is the Fréchet distance with an additional continuity requirement: the leash is not allowed to switch discontinuously, e.g. jump over obstacles.

2 Discretization Using Steiner Points

To approximate $\delta_F(P, Q)$, we discretize the problem by placing Steiner points along edges of R.

For each vertex $v \in R$, the vertex radius for v is defined as $r(v) = (\epsilon B)/(n w_{max}(v))$, where ϵ is a positive real number defining the quality of the approximation, B is a lower bound on $\delta_F(P, Q)$, and $w_{max}(v)$ is the maximum weight among all weighted regions incident to v. The disk of radius $r(v)$ centered at v defines the vertex vicinity of v. No Steiner points are placed inside the vertex vicinity of v. B can be computed in $O(pq \log(pq))$ time using the standard (unweighted case) continuous Fréchet distance algorithm described in [2], where p and q are the number of vertices of P and Q, respectively.

We apply a discretization scheme similar to that in [1] (other schemes, extending from previous work, are also possible) to place Steiner points along edges forming a geometric progression. For any point v on an edge e in R, let $\overline{E(v)}$ be the set of edges in R not incident to v and let $d(v)$ be the minimum distance between v and edges in $\overline{E(v)}$. It follows from [1] that the number of Steiner points placed on an edge is $O(C(e) 1/\epsilon \log 1/\epsilon)$, where $C(e) = O(\frac{|e|}{d(e)} \log \frac{|e|}{\sqrt{r(v_1) r(v_2)}})$, $|e|$ is the Euclidean length of e, and $d(e) = \sup\{d(v) | v \in e\}$. Let N denote the total number of Steiner points, including the vertices of R. We have $N = O((1/\epsilon) \log(1/\epsilon) n \log n)$. We refer to a line segment bounded by two consecutive Steiner points as a *Steiner edge*. An *hourglass* is the union of all links intersecting with the same sequence of Steiner edges in the same order. Let H be an hourglass defined by a sequence of Steiner edges $\{e_1, e_2, \ldots, e_k\}$, where $e_1 \in P$ and $e_k \in Q$.

Lemma 1 *Let l and l' be two segments in H. Then, $S(l) \leq (1 + 2\epsilon) S(l') + 2\epsilon B$.*

Proof. Omitted. □

Let l_H be an arbitrary segment in H, with endpoints on P and Q. Let α and β be two reparametrizations that define a matching between P and Q. Let $J = \{H_1, H_2, \ldots, H_k\}$ be the set of hourglasses that are traversed by the leash $P(\alpha(r))Q(\beta(r))$. For an hourglass $H \in J$, let $I_H = \{e | e = P(\alpha(r))Q(\beta(r)), r \in [0, 1], e \in H\}$. Let $H(\alpha, \beta)$ be the segment in I_H with the largest weighted length. That is, $S(H(\alpha, \beta)) = \max_{e \in I_H} S(e)$. Let $\delta(\alpha, \beta) = \sup_{r \in [0,1]} S(P(\alpha(r)), Q(\beta(r)))$.

Lemma 2 $|\max_{H \in J} S(l_H) - \delta(\alpha, \beta)| \leq 4\epsilon \delta(\alpha, \beta)$.

Proof. Omitted. □

We say that $\delta(J) = \max_{H \in J} S(l_H)$ is a 4ϵ-approximation of $\delta(\alpha, \beta)$. Given a sequence of hourglasses $J = \{H_1, H_2, \ldots, H_k\}$, we call J *legal* if there exist two reparametrizations α, β that define a leash traversing the same sequence of hourglasses as J.

Lemma 3 *There exists a legal sequence of hourglasses $\hat{J}$ such that $\delta(\hat{J})$ satisfies $|\delta(\hat{J}) - \delta_F(P, Q)| \leq 4\epsilon \delta_F(P, Q)$, that is, $\delta(\hat{J})$ is a 4ϵ-approximation of $\delta_F(P, Q)$.*

Proof. (Sketch) Let $\hat{\alpha}$, $\hat{\beta}$ be the optimal reparametrizations that give the Fréchet distance between P and Q. Let $\hat{J}$ be the sequence of hourglasses traversed by the leash defined by $\hat{\alpha}$ and $\hat{\beta}$. Applying *Lemma* 2, we have, $|\delta(\hat{J}) - \delta(\hat{\alpha}, \hat{\beta})| = |\delta(\hat{J}) - \delta_F(P, Q)| \leq 4\epsilon \delta(\hat{\alpha}, \hat{\beta}) = 4\epsilon \delta_F(P, Q)$. □

3 Fréchet Distance Between Two Polygonal Chains

We attack the problem in the parameter space $D = [0, 1]^2$, where a leash $P(s)Q(t)$ is associated to a point $(s, t) \in D$. We refer to $(s, t) \in D$ as the *dual point* of the leash $P(s)Q(t)$.

Lemma 4 *All leashes through a Steiner point v correspond to a curve in D, with equation $C_v : st + c_1 s + c_2 t + c_3 = 0$, where c_1, c_2, and c_3 are constants.*

Proof. Let $v = (a, b)$, $P(0) = (x_1, y_1)$, $P(1) = (x_2, y_2)$, $Q(0) = (x_3, y_3)$, $Q(1) = (x_4, y_4)$. We have

$$P(s) = (x_1 + (x_2 - x_1)s, y_1 + (y_2 - y_1)s),$$

$$Q(t) = (x_3 + (x_4 - x_3)t, y_3 + (y_4 - y_3)t),$$

and $P(s)Q(t) : y - (y_1 + (y_2 - y_1)s) = \frac{y_1+(y_2-y_1)s-(y_3+(y_4-y_3)t)}{x_1+(x_2-x_1)s-(x_3+(x_4-x_3)t)}(x - (x_1 + (x_2 - x_1)s))$. Since $P(s)Q(t)$ passes through v, we obtain that $(b-y_1-(y_2-y_1)s)(x_1-x_3+(x_2-x_1)s-(x_4-x_3)t)) = (a-x_1-(x_2-x_1)s)(y_1-y_3+(y_2-y_1)s-(y_4-y_3)t)$, and thus we have

$$C_v : st + c_1 s + c_2 t + c_3 = 0,$$

where c_1, c_2, and c_3 are constants. □

We call the curve C_v the *dual curve* of Steiner point v. Next we define the relative position of two leashes with respect to a Steiner point v. Given two leashes $P(s)Q(t)$ and $P(s')Q(t')$, we say they are on the *same side* of v if and only if there exists two continuous functions $\alpha', \beta' : [0, 1] \to [0, 1]$, such that $\alpha'(0) = s$, $\alpha'(1) = s'$, $\beta'(0) = t$, $\beta'(1) = t'$ and $v \notin P(\alpha'(r)Q(\beta'(r)), \forall r \in [0, 1]$. Note that α', β' are not necessarily monotone. Intuitively, two leashes are on the same side of v if there exists a walk from one leash to the other one without crossing v, i.e. one leash can reach the other one by sweeping its end points on their respective curves without crossing v.

Lemma 5 *C_v divides D into two partitions, each partition corresponding to all links on the same side of point v.*

Proof. Given two links $P(s)Q(t)$ and $P(s')Q(t')$, since there is a one-to-one correspondence between all possible walks from $P(s)Q(t)$ to $P(s')Q(t')$ and all paths from (s, t) to (s', t') in D, if (s, t), (s', t') are on different sides of C_v, all paths between (s, t) and (s', t') must pass C_v at least once. That is, no walk exists between the two links without crossing v. □

Hence, we can partition D by the dual curves of the Steiner points, such that each cell corresponds to an hourglass. Using the algorithm in [3], the partition can be computed in $O(N \log N + k)$ time and $O(N + k)$ space, where $N = 1/\epsilon \log(1/\epsilon)n \log n$ is the total number of Steiner points and k is the number of cells, which is $O(N^2)$ in the worst case. Let A denote the partition.

The sequence of cells traversed by a monotone path in D corresponds to a legal sequence of hourglasses traversed by a leash following a match of P and Q, and vice versa. The Fréchet distance between P and Q can then be approximated as follows:

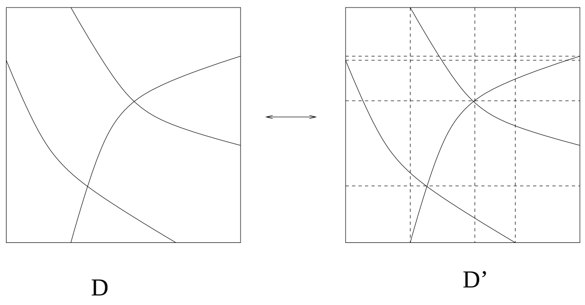


Figure 2: An example of the decomposition of D.

1. Place Steiner points as described earlier.
2. Partition D by the dual curves of the Steiner points.
3. For each cell Z, choose an arbitrary leash l and assign $S(l)$ as the weight of the cell.
4. Find a monotone path T in D, from its left bottom corner, $(0, 0)$, to its top right corner, $(1, 1)$, such that the maximum weight of the cells traversed by T is minimized.

By Lemma 3 the maximum weight of the cells traversed by T is a 4ϵ-approximation of the Fréchet distance between P and Q.

To find an optimal path T in D, we decompose D by extending a horizontal as well as a vertical line from every vertex until it reaches the boundary of D. See Fig. 2 for an illustration. Let the new subdivision be D'.

Lemma 6 *Given an edge e in D', let p be a point on e and let T_p be an arbitrary monotone path from $(0, 0)$, i.e. the bottom left corner of D', to p. Then, there exists a monotone path from $(0, 0)$ to any point on e with the same cost as T_p.*

Proof. We add e to D. In D, we extend a vertical line as well as a horizontal line from the end points of e. Let L_1, L_2 be the two horizontal lines which define a horizontal *slab* and let L_3, L_4 be the two vertical lines which define a vertical slab. Obviously, T_p has the same cost in D as it has in D'. If T_v enters the horizontal slab before it enters the vertical slab, let v be the entry point. Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of curves on edges of D, which are bounded by the horizontal slab and traversed by T_p. Since the horizontal slab does not contain any vertex of D in its interior, no two curves in S intersect with each other. Hence we can construct a monotone path from v to any point on e that traverses S, i.e. we can construct a monotone path from $(0, 0)$ to any point on e such that it has the same cost as T_p. See Fig. 3 for an illustration. The same argument holds true if T_p enters the vertical slab first. □

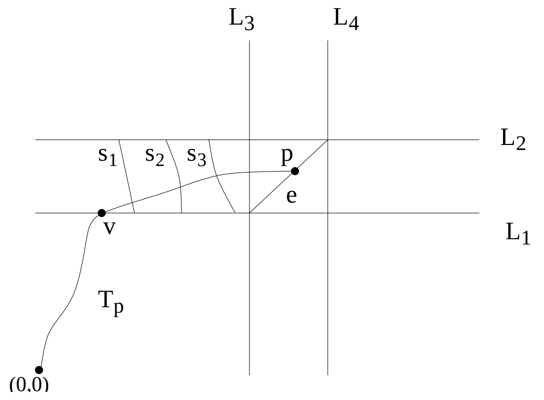


Figure 3: There exists a monotone path from $(0, 0)$ to every point in e, which has the same cost as T_p.

Thus, given an optimal monotone path T_p from $(0, 0)$ to an arbitrary point $p \in e$, we can construct a monotone path from $(0, 0)$ to any point in e, which has the same cost as T_p and is also optimal. To find the optimal monotone path from $(0, 0)$ to $(1, 1)$, we construct a directed graph having the (open) edges as well as the vertices of D' as vertices. In the graph, a direct edge is added from a node v_1 to a node v_2 if, in D', v_1 and v_2 share a common cell and there exists a monotone path from v_1 to v_2. The shorst path can be found by running Dijkstra's algorithm on the directed graph.

Time complexity: The complexity of D' is N^4, where $N = (1/\epsilon)\log(1/\epsilon)n\log n$ is the number of Steiner points used, since each cell in D' has $O(1)$ edges and vertices, and thus each cell contributes $O(1)$ edges to the directed graph. Then, the directed graph has $O(N^4)$ vertices and $O(N^4)$ edges and Dijkstra's algorithm takes $O(N^4 \log N)$ time.

References

[1] L. Aleksandrov, A. A. Maheshwari, and J.R. Sack. Approximation algorithms for geometric shortest path problems. *In Proc. 32nd Annual ACM Symposium on Theory of Computing*, pp. 286-295, 2000.

[2] H. Alt and M. Godau. Computation the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75-91, 1995.

[3] N.M. Amato, M.T. Goodrich, and E.A. Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. *In Proc. 11th Annual CAM-SIAM Symposium on Discrete Algorithms*, pp. 705-706, 2000.

[4] K. Buchin, M. Buchin and C. Wenk. Computing the Fréchet distance between simple polygons in polynomial time. *In Proc. 22rd Symposium on Computational Geometry*, pp. 80-87, 2006.

[5] E.W. Chambers, É. Coline De Verdière, J. Erickson, S. Lazard, F. Lazarus and S. Thite. Walking your dog in the woods in polynomial time. *In. Proc. 24th Annual ACM Symposium on Computational Geometry*, pp. 101-109, 2008.

[6] A.F. Cook IV and C. Wenk. Geodesic Fréchet Distance Inside A Simple polygon. *In Proc. 25th International Symposium on Theoretical Aspects of Computer Science*, pp. 193-204, 2008.

[7] O. Daescu and J. Palmer. Minimum Separation in Weighted Subdivisions. To appear in *International Journal of Computational Geometry and Applications*

[8] T. Eiter and H. Mannila. Computing discrete Fréchet distance. *Technical Report CD-TR 94/64*, Information Systems Department, Technical University of Vienna, 1994.

[9] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Mathematico di Palermo*, 22:1-74, 1906.

[10] A. Maheshwari and J. Yi. On computing Fréchet distance of two paths on a convex polyhedron. *In Proc. 21st European Workshop on computational Geometry*, pp. 41-44, 2005.

[11] G. Rote. Computing the Fréchet distance between piecewise smooth curves. *Technical Report*, ECGTR-241108-01, 2005.

[12] G.T. Toussaint. Solving geometric problems with the 'rotating calipers. *In Proc. 2nd IEEE Mediterranean Electrotechnical Conference (MELECON '83)*, pp. A10.02/1-4, 1983.

[13] K. Buchin, M. Buchin, and Y. Wang. Exact Partial Curve Matching under the Fréchet Distance. To appear *In Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2009.

Safe Routes on a Street Graph with Minimum Power Transmission Range

Manuel Abellanas[*,†] Antonio L. Bajuelos[‡,§] Inês Matos[‡,¶]

Abstract

Let S be a set of n points in the plane (antennas). An object is said to be 2-covered with range r if every point of such object is interior to at least two discs (not necessarily the same) centered at S of radius r. The following problem is considered in this paper: given a set S of n antennas and a planar geometric graph $G = (N, E)$, calculate the minimum power transmission range of S so that a 2-covered path between two given nodes of G exists. Is is described an algorithm to solve this problem in two phases. In the first phase (*preprocessing phase*), graph G is transformed into a weighted graph G_w (using the second order Voronoi diagram of S) and then a minimum spanning tree of G_w, T_w, is found. This phase takes $\mathcal{O}(|E| \times n)$, $|E| > \log n$, time. In the second phase (*solution phase*), the minimum power transmission range of S and a 2-covered path are calculated using T_w. Regarding time complexity, this second phase is linear on the number of edges of T_w.

1 Introduction and Related Works

Let S be a set of n points in the plane that represent the location of n antennas (or any device able to send or receive some sort of signal). Suppose all antennas have the same power transmission range $r \in \mathbb{R}^+$ that is variable. The distance between a point q and a set S of points is the shortest distance between q and every point of S. The minimum range of the antennas so that S covers q is exactly the distance between q and S. A point covered by two or more antennas is said to be 2-*covered* by S. The concept of 2-covering is useful to assure that the points remain covered when one antenna fails. Let D be the set of discs centered at the antennas of S of radius r. Each nonempty intersection between two discs of D is called a lens. It is easy to see that 2-covered regions result from the union of these lenses. Therefore, a point is 2-covered by S if it belongs to a lens (see Figure 1). The antennas' covering range, r, depends directly on their transmission power which, on its turn, is responsible for the costs associated to the service they provide. For that reason, it is important to minimize the value of r.

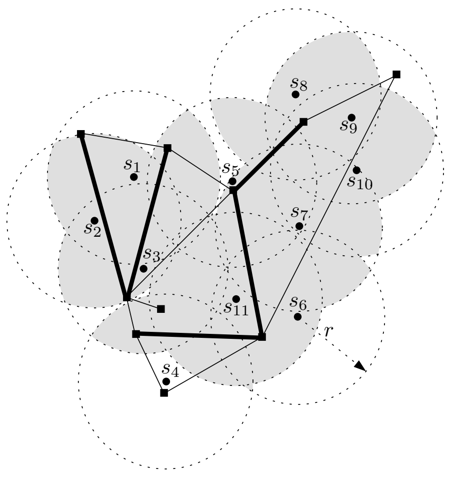


Figure 1: Set S is the set $\{s_1, s_2, \ldots, s_{11}\}$. Graph's edges that are 2-covered by the antennas of S are shown in a heavier trace.

Consider now a connected geometric planar graph G together with set S. Suppose that the edges of G represent streets/roads and its nodes represent reachable locations using those streets (see Figure 1). A path on G using only the edges that are 2-covered by S is called a 2-covered path or a 2-path. Given two nodes n_i and n_j of G, our main goal is to compute the minimum power transmission range so that there is a 2-path on G connecting n_i and n_j.

Related Works: in [1] the authors study a problem equivalent to the one here proposed and other multi-parametric optimization problems for 1-covering. In [2], a subset of a given set of discs with variable radii whose costs depend on their radii is computed, as well as the same subset but with fixed costs to cover a given line segment at minimum cost.

This paper is structured as follows. The next section is divided in two subsections. In the first subsection, graph G is preprocessed and converted into

*Facultad de Informática, Universidad Politécnica de Madrid, mabellanas@fi.upm.es

†Co-supported by Project Consolider Ingenio 2010 i-MATH C3-0159 and MICINN Project MTM2008-05043

‡Departamento de Matemática & CEOC, Universidade de Aveiro, {leslie,ipmatos}@ua.pt

§Supported by CEOC through *Programa* POCTI, FCT, co-financed by EC fund FEDER

¶Supported by a FCT fellowship, grant SFRH/BD/28652/2006 and by CEOC through *Programa* POCTI, FCT, co-financed by EC fund FEDER

a weighted graph G_w. In this subsection, a minimum spanning tree (MST) of G_w is also found. Such MST is later used in the second subsection to calculate the minimum power transmission range of S to guarantee the existence of a 2-path between two nodes of G. Finally, conclusions and some remarks are presented in Section 3 .

2 Minimum 2-Covering of a Path between two nodes of a Street Graph

Let S be a set of n antennas. The second order Voronoi diagram of S, $\mathrm{VD}_2(S)$, divides the plane into several regions by grouping points that share the same two closest antennas [3]. This proximity structure is naturally related to this problem since a point is 2-covered if it is interior to the lens defined by its two closest antennas. For this reason, a point q is 2-covered if the power transmission range of the two closest antennas to q is enough to reach q. The minimum power transmission range of S that 2-covers an object x is denoted by $\mathrm{MR}_S(x)$.

Let $G = (N, E)$ be a connected geometric planar graph whose nodes represent locations and its edges represent roads/streets that connect such locations. Without loss of generality, it is assumed that $|E| > \log n$. Given two nodes n_i and n_j of G, the smallest range of S which ensures the existence of a 2-path between n_i and n_j on G is denoted by $\mathrm{MR}_{S,G}(n_i, n_j)$. To simplify the notation, it can also be denoted by $\mathrm{MR}_S(n_i, n_j)$ if G is clear from the context.

2.1 First Phase: Preprocess

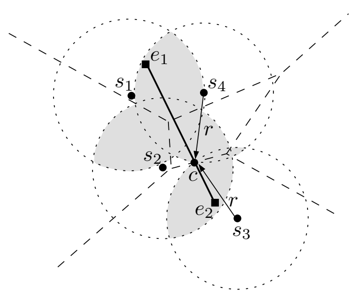


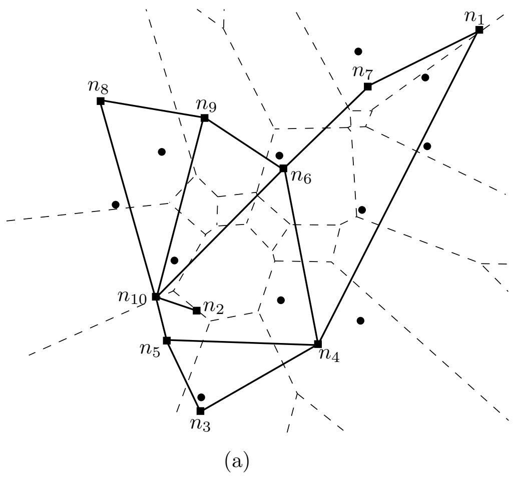
Figure 2: The minimum power transmission range r of S to 2-cover $\overline{e_1 e_2}$ is $r = d(c, s_3) = d(c, s_4)$. $\mathrm{VD}_2(S)$ is shown in a dashed trace.

Given a line segment e, $\mathrm{MR}_S(e)$ is calculated using the intersection points between e and $\mathrm{VD}_2(S)$, $I_e = \{e\} \cap \mathrm{VD}_2(S)$. It is easy to see that the minimum power transmission range needed to 2-cover every point of I_e and the extreme points of e is $\mathrm{MR}_S(e)$ (see Figure 2).

The procedure explained in this subsection acts as a preprocess that transforms graph G into weighted graph G_w. To make this transformation, $\mathrm{MR}_S(e)$ is assigned as the weight of each edge e of G (see Figure 3).

Proposition 1 *Let $G = (N, E)$ be a graph and S a set of n antennas. The $\mathrm{MR}_S(e)$ for every edge $e \in E$ can be calculated in $\mathcal{O}(|E| \times n)$ time.*

Proof. Computing $\mathrm{VD}_2(S)$ takes $\mathcal{O}(n \log n)$ since there are n antennas [6]. Then using $\mathrm{VD}_2(S)$, it is possible to find the set $I_e = \{e\} \cap \mathrm{VD}_2(S)$ for every edge $e \in E$ in $\mathcal{O}(n \log n + |E| \times n)$ time. Next there is the need to add the extreme points of e to the set I_e. For each intersection point $p \in I_e$, $\mathrm{MR}_S(p)$ is calculated in constant time using $\mathrm{VD}_2(S)$ as it is the distance between p and its second closest antenna. Therefore, calculating $\mathrm{MR}_S(e) = \max\{\mathrm{MR}_S(p), \forall p \in I_e\}$ is a lin-



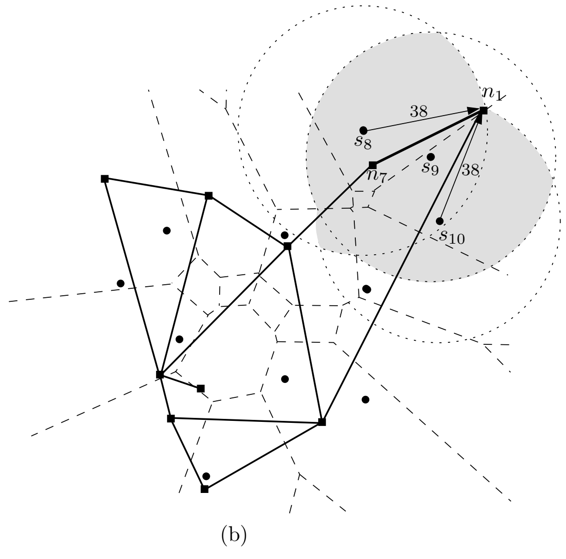


Figure 3: Set S of eleven antennas represented by dots. $\mathrm{VD}_2(S)$ is shown in a dashed trace. (a) Graph G whose nodes are represented by squares. (b) The graph's edge connecting n_1 and n_7 has weight 38 which is the minimum power transmission range of S to 2-cover such edge.

ear procedure for each edge e. As it is supposed that $|E| > \log n$, $\mathrm{MR}_S(e)$ for every edge $e \in E$ can be computed in $\mathcal{O}(|E| \times n)$ time. □

Let us remark that a minimum spanning tree of G_w, T_w, can be found in linear time using an algorithm by Matsui [4] since G_w is a planar graph. As it is shown in the next subsection, T_w eases the calculations since it stores important information concerning the antennas' arrangement in relation to G_w. To summarize, the algorithm can be described as follows.

```
ALGORITHM: PREPROCESSING PHASE
In: Set $S$ of $n$ antennas and a street graph $G$
Out: Graph $G_w$ and $T_w$, a MST of $G_w$
  1. Compute $\mathrm{VD}_2(S)$, the Second Order Voronoi Diagram of $S$.
  2. Convert $G$ into weighted graph $G_w$:
     For every $e \in G$ do
        (a) Calculate $\mathrm{MR}_S(e)$
        (b) $w(e) \leftarrow \mathrm{MR}_S(e)$
  3. Find $T_w$, a MST of $G_w$.
```

The next result is a direct consequence of Proposition 1.

Theorem 2 *Given a set S of n antennas and graph $G = (N, E)$, computing a weighted graph G_w and a* MST *of G_w can be done in $\mathcal{O}(|E| \times n)$ time.*

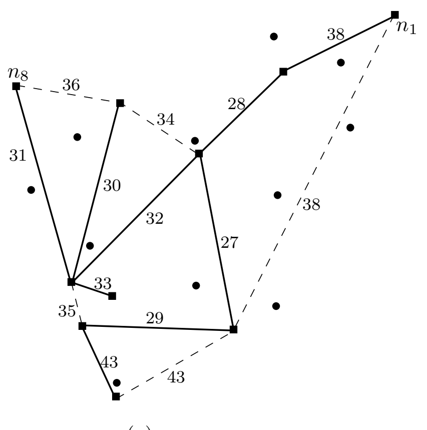
2.2 Second Phase: Solution

Given two nodes n_i and n_j of G_w, the algorithm in this subsection shows how to calculate $\mathrm{MR}_{S,G_w}(n_i, n_j)$ and how to find a 2-covered path on G_w connecting n_i and n_j.

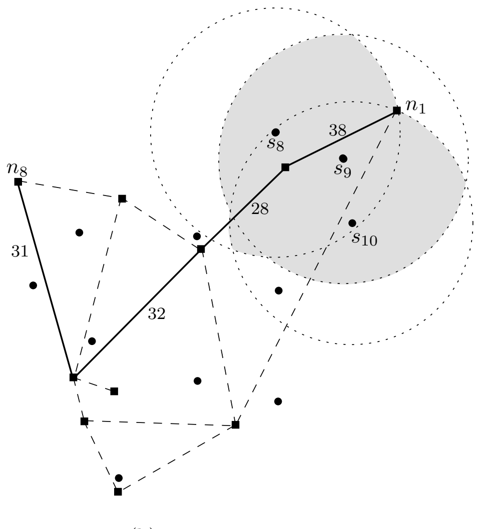
Let P be a 2-covered path between two nodes on G_w. It is easy to see that the minimum power transmission range that ensures the existence of P is given by the weight of the heaviest edge of P. The next property shows that it is only necessary to consider the edges of a MST of G_w to find a 2-path between two given nodes of G_w.

Proposition 3 *Let G_w be an edge-weighted connected graph. For any path P on G_w, let us consider the weight of P as the weight of its heaviest edge. Then, for every pair of nodes of G_w, the path connecting them on a MST of G_w is a minimum weight path between such pair.*

Proof. Let G_w be an edge-weighted connected graph and T_w a minimum spanning tree (MST) of G_w. Suppose that P is the only path on T_w connecting nodes n_i and n_j and e is its heaviest edge. Consequently, P has weight $w(e)$. Now suppose that path P^* on G_w is a minimum weight path connecting n_i and n_j. Its weight is given by e^*, its heaviest edge, and so $w(e*) < w(e)$. Since P is heavier than P^*, the edge e is not part of P^*. If paths P and P^* are united, then a cycle is created. That cycle contains e which clearly is its heaviest edge. But that contradicts the hypothesis that e is an edge of a MST of G_w. Therefore, a minimum weight path between two nodes of G_w is the path on T_w connecting those nodes. □



(a)



(b)

Figure 4: (a) A MST of a weighted graph is shown in a solid trace. (b) The path connecting n_1 and n_8 on the tree only exists if the antennas' power transmission range is at least $d(s_8, n_1) = d(s_{10}, n_1) = 38$.

It is now clear that a 2-path between two nodes on a weighted graph with minimum power transmission range can be computed using a MST of such graph (see Figure 4(a)). Working with a MST is easier since a path between two nodes is unique. Follows the core of the algorithm to find a 2-path connecting two nodes of G_w with minimum range, taking advantage of a

MST of G_w.

```
ALGORITHM: SOLUTION PHASE
In: $T_w$, a MST of $G_w$
Out: $P$, a 2-covered path and $\mathrm{MR}_{S,G_w}(n_i, n_j)$
    1. Use the algorithm DFS [5] to find a path
       $P$ on $T_w$ between $n_i$ and $n_j$.
    2. $\mathrm{MR}_{S,G_w}(n_i, n_j) \leftarrow \max\{w(e) : \forall e \in P\}$.
```

In Figure 4(b) there is an example of a path on a MST connecting nodes n_1 and n_8 with minimum power transmission range. If all the antennas have range $r = \max\{31, 32, 28, 38\} = 38$, the path is 2-covered with minimum power transmission range. The largest range is the one needed to 2-cover n_1. It is given by the distance between n_1 and s_8 (or s_{10}), such distance is exactly $\mathrm{MR}_{S,G_w}(n_1, n_8)$.

Theorem 4 *Given a set S of n antennas, let n_i and n_j be two nodes of graph G_w and $T_w = (N, B)$ a MST of G_w. Then a 2-covered path between n_i and n_j on G_w with range $\mathrm{MR}_{S,G_w}(n_i, n_j)$ can be computed on T_w in $\mathcal{O}(|B|)$ time.*

3 Concluding Remarks

The concept of 2-covering was introduced in this paper, as well as a related problem: calculating the minimum power transmission range to ensure the existence of a 2-covered path between two nodes of a graph. Given a set S of n antennas, the solution to this problem is divided in two phases. In the first phase, graph $G = (N, E)$ is converted into a weighted graph using a $\mathcal{O}(|E| \times n)$, $|E| > \log n$, time algorithm. In the second phase, a 2-path between two nodes of G is found, as well as the minimum power transmission range of S to guarantee the existence of such path. This last algorithm is linear on the number of edges of a minimum spanning tree of the weighted graph.

References

[1] M. Abellanas and G. Hernández. *Optimización de Rutas de Evacuación.* Proc. of the XII Encuentros de Geometría Computacional, 2007, Valladolid, Spain, 273–280.

[2] A. Agnetis, E. Grande, P.B. Mirchandani, P.B. and A. Pacifici. *Covering a Line Segment with Variable Radius Discs.* To appear on Computers and Operations Research, 36, 5, 2009, 1423–1436.

[3] F. Aurenhammer and R. Klein. *Voronoi diagrams.* In J.-R. Sack, J. Urrutia (eds.), *Handbook of Computational Geometry*, Elsevier, 2000, Amsterdam, 201–290.

[4] T. Matsui. *The minimum spanning tree problem on a planar graph.* Discrete Applied Mathematicss, 58, 1, 1995, 91–94.

[5] S.W. Golomb and L.D. Baumert. *Backtrack Programming.* Journal of ACM, 12, 4, 1965, 516–524.

[6] D.T. Lee. *On k-Nearest Neighbor Voronoi Diagrams in the Plane.* IEEE Transactions on Computers, 31, 6, 1982, 478–487.

Line Segment Facility Location in Weighted Regions [*]

Yam Ki Cheung[†] and Ovidiu Daescu[†]

1 Introduction

In this paper we consider a geometric optimization problem that we call the *line segment facility location in weighted regions*. We study the problem in a weighted subdivision $R = \{R_1, R_2, \ldots, R_{n'}\}$ of the plane with n vertices, with each region $R_i \in R$ having associated a positive integer weight w_i. Without loss of generality, we assume that R is triangulated. The weighted length of a segment st within a region $R_i \in R$ is defined as $w_i|st|$, where $|st|$ is the Euclidean length of st. We denote by $S(st)$ the weighted length of a segment $st \in R$. The weighted length of st $S(st) = \sum_{st \cap R_j \neq \emptyset} w_j * d_j(st)$, where $d_j(st)$ is the Euclidean length of st within region R_j.

The problem is defined as follows: Given a set of l points $P = \{s_1, s_2, \ldots, s_l\} \in R$, find a facility L, which is a line segment, such that each point in P is connected to L via an orthogonal link and some *cost* $C(L)$ associated with the facility location is minimized (see Fig. 1 for an example). We consider two versions of this problem, depending on how $C(L)$ is defined. For the first version (P_1), $C(L) = S(L)$. That is, we want to minimize the weighted length of L. For the second version (P_2), $C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$, where L_i is the orthogonal link from s_i to L. That is, we want to minimize the sum of weighted length of all orthogonal links and L.

The facility location problem is a well-studied problem in operations research and computer science literature [5, 6, 7, 3]. The problem has many formulations, resulting in various definitions for the objective function. In our formulation, there is a cost for opening a facility and also for constructing orthogonal links that connect customers to this facility. The goal of P_1 is to minimize the cost of building the line facility, while P_2 takes the cost of building the facility as well as the cost of building orthogonal links into account. One may imagine that the facility L is an oil pipeline and the points in P are oil wells. The oil field is divided into weighted regions based on its characteristics (cost of digging, ownership rights, etc.) From each well, a spur pipeline is to be connected directly to L, in straight line. Given the x- and y-coordinates of the wells, the goal is to find the optimal location for the main pipeline that minimizes the total cost. In unweighted environment, Megiddo and Tamir [7] solved the problem of minimizing the sum of the Euclidean length of orthogonal links more than two decades ago. They showed that the optimal line facility can be found in $O(n^2 \log n)$ time by proving that it must pass through at least two of the points in P. Later, Imai et. al. [5] proved that in L_1-metric the optimal solution can be computed in linear time. Cheung and Daescu [3] showed that the weighted version of this problem, i.e. $C(L) = \sum_{i=1}^{l} S(L_i)$, can be solved by dividing the problem into a number ($O(l^2n^2)$ in worst case) of 1-variable subproblems. The objective function of each subproblem has the form

$$\sum_{i=1}^{l} S(L_i) = \sqrt{1+m_o^2}\left(\sum_{j=1}^{M} \frac{c_j}{m_o - m_j} + \sum_{i=1}^{l} \frac{b_i m_o + a_i}{m_o^2+1} + C\right),$$

for some constants a_i, b_i, c_i, C, where l is the number of points in P, n is the number of vertices of R and m_o is the slope of orthogonal links.

The proposed problem is also closely related to the *optimal weighted link problem*, in which the goal is to minimize the weighted length of a line segment st connecting two given regions, R_s and R_t, of a weighted subdivision R. In [2], it has been proven that the problem to minimize the cost $S(st) = \sum_{st \cap R_j \neq \emptyset} w_j * d_j(st)$ can be reduced to $O(n^2)$ global optimization problems, each of which asks to minimize a 2-variable function over a convex domain. In [4], it has been shown that to minimize the objective function $S(st)$, st must pass through a vertex of R.

Results: For P_1, we show that the objective function $C(L)$ can be expressed in the form of a summation of a number of fractional terms, that resembles those

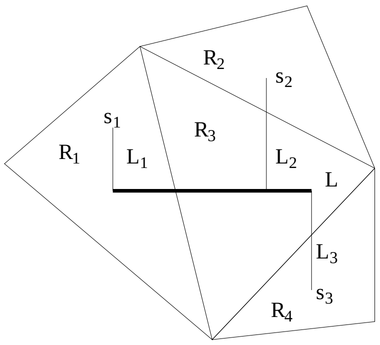


Figure 1: A line facility L and the orthogonal links from $P = \{s_1, s_2, s_3\}$ to L

[*]This work was supported in part by NSF award CCF-0635013.

[†]Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA, Email: {ykcheung,daescu}@utdallas.edu.

in [2, 4]. The problem can be reduced to a number of ($O(l^2n^2)$ in worst case) 1-variable optimization problems, which can be solved by either computing exact roots of polynomials or by approximation algorithms. Using point-line duality transforms the feasible domain for each subproblem is an arc or a line segment on the dual plane. Then, we show that P_2 can be solved in a similar fashion.

2 P_1: minimize the weighted length of L

In this section, we consider the problem of minimizing the weighted length of L, i.e. $C(L) = S(L)$, such that the points in P can access L via orthogonal links. We first derive the general objective function. Then, we show that this optimization problem can be reduced to a number of 1-variable subproblems and give a prune-and-search approximation algorithm for solving the subproblem.

2.1 The rotating calipers

Obviously, we want to set the line facility L as short as possible, while each point in P is able to connect to L via orthogonal links. Let $CH(P)$ denote the convex hull of P. A line e is a *line of support* of $CH(P)$ if e is tangent to $CH(P)$ The length of L is minimized when L is a segment bounded by two parallel lines of support enclosing $CH(P)$ and orthogonal to L. See Fig. 2 for an illustration. A pair of vertices of $CH(P)$, $\{p_i, p_j\}$, is an *antipodal pair* if it admits two parallel lines of support enclosing $CH(P)$. Applying the *rotating calipers* procedure [8], we can generate all possible antipodal pairs (h pairs in the worst case) for $O(h)$ slope intervals in $O(h)$ time, where h is the number of vertices of $CH(P)$.

2.2 Optimization of $S(L)$

Given a slope m, let the antipodal pair that admits the two parallel lines of support of slope $-1/m$ be $p_i = (a, b)$ and $p_j = (c, d)$, respectively. Also, let L be a line segment with end points on the two parallel lines of support and orthogonal to them. Let the sequence of edges intersected by L be $Seq(L) = (e_1, e_2, \ldots, e_k)$, where $k = O(n)$, $e_j : y = m_jx + p_j$, for $j = 1, 2, \ldots, k$, and e_1 and e_k are the two parallel lines of support. We have,
$S(L) = \sqrt{1+m^2}\sum_{j=1}^{k-1} w_i|x_{j+1} - x_j|$
$= \sqrt{1+m^2}\sum_{j=1}^{k-1} w_j(\frac{p_{j+1}-p}{m-m_{j+1}} - \frac{p_j-p}{m-m_j})$,
where m and p are the slope and intercept of L respectively, x_j is the x-coordinate of the intersection between L and e_j, and w_j is the weight of the region bounded by edges e_j and e_{j+1}. Since e_1, e_k are the two parallel lines of support, we have $e_1 : y = -1/m(x - a) + b$ and $e_k : y = -1/m(x - c) + d$. It follows that, $S(L) = \sqrt{1+m^2}(\sum_{j=2}^{k-2} w_j(\frac{p_{j+1}-p}{m-m_{j+1}} - \frac{p_j-p}{m-m_j}) +$

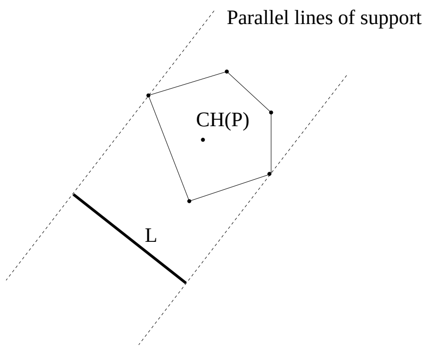


Figure 2: L is bounded by two parallel lines of support enclosing $CH(P)$.

$w_1(\frac{p_2-p}{m-m_2} - \frac{b+a/m-p}{m+1/m}) + w_{k-1}(\frac{d+c/m-p}{m+1/m} - \frac{p_{k-1}-p}{m-m_{k-1}}))$
$= \sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c_j+d_jp}{m-m_j} + \frac{Cmp+Dm+E}{1+m^2})$,
where C, D, E, c_j and d_j are all constants.

Next, we analyze the problem in the dual space. We use the duality transform which maps a line $y = mx + p$ on the plane onto a point (m, p) in the dual plane and maps a point (a, b) onto the line $y = -ax + b$ on the dual plane. Note that there exist a one-to-one correspondence between L and the line supporting it. Observe that if we change the slope m and intercept p of L, $seq(L)$ changes, i.e. the expression of the objective function $C(L) = S(L)$ changes, if
(1) L sweeps through a vertex of R, or
(2) L sweeps through an intersection between a line of support and an edge of R, or
(3) L rotates through a slope such that the antipodal pair changes.

We introduce three sets of curves or lines to properly partition the dual space such that each cell corresponds to all segments in the primal space which have the same functional expression for the weighted length. First, we transform all vertices of R to the dual space. Let A_v denote this arrangement, which consists of n lines. Next, consider that L intersects a line of support e' at an edge e. As the line of support e' rotates, the intersection between L and e' moves along edge e. We can show that the transform of all lines intersecting with e' on e and orthogonal to e' yields a curve in the dual space in the form

$$p = -\frac{(b_i - p_e)m^2 + (a_i - m_eb_i)m - m_ea_i - p_e}{m_em + 1},$$

where m_e and p_e are the slope and intercept of e respectively and a_i and b_i are constants. Since there are $O(h)$ antipodal pairs of vertices and $O(n)$ edges of R, we have a total of $O(hn)$ such curves. Let A_c' denotes the arrangement of these curves. Finally, let $M = \{m_1, m_2, \ldots, m_h\}$ be the set of slopes of edges in $CH(P)$ and let $M' = \{-1/m : m \in M\}$. Recall that in the rotating calipers procedure, the antipodal pair changes only when the calipers are rotating through the slopes in M, or equivalently when L rotates through the slopes in M', since L is orthogonal to the calipers. Let A'_r be the arrangement of $O(h)$ vertical lines $\{x^* = m : m \in M'\}$ in the dual space.

Lemma 1 *$A_v \cup A'_c \cup A'_r$ partitions the the dual space such that each cell corresponds to all segments in the primal space which have the same functional expression for the weighted length.*

Proof. Let L and L' be two line facilities such that $S(L)$ and $S(L')$ have different functional expressions for the weighted length. Due to order preserving property of the dual transform, their corresponding dual points L^* and L'^* must be separated by at least one line or curve in A_v, A'_c or A'_r. The proof follows. □

Using the algorithm in [1], the arrangement $A_v \cup A'_c \cup A'_r$ can be computed in $O(nh\log(nh) + k)$ time and $O(nh + k)$ space, where k is the number cells, which is $O(n^2h^2)$ in worst case.

Lemma 2 *The global minimum of $C(L)$ can be found at non-vertical boundaries of the partition, i.e. on the lines or the curves in $A_v \cup A'_c$.*

Proof. Observe that when the slope m of L is fixed, the objective function $C(L)$ is linear and monotonic with respect to the intercept p of L. The minimum of $C(L)$ cannot be found in the interior of the partition. □

Lemma 3 *P_1 can be reduced to $O(n^2h^2)$ 1-variable subproblems. The domain for each subproblem is a piece of non-vertical boundary on the partition $A_v \cup A'_c \cup A'_r$.*

Proof. The complexity of the overall partition $A_v \cup A'_c \cup A'_r$ is $O(n^2h^2)$. The result follows from *Lemma* 1 and *Lemma* 2. □

If the domain of the subproblem is on A_v, L passes through a vertex. Let $v : (x_i, y_i) \in R$ be the vertex passed by L. We have,
$L : y - y_i = m(x - x_i)$,
$p = y_i - mx_i$,
$S(L) = \sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c_j + d_j p}{m - m_j} + \frac{Cmp + Dm + E}{1+m^2})$
$= \sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c_j + d_j(y_i - mx_i)}{m - m_j} + \frac{Cm(y_i - mx_i) + Dm + E}{1+m^2})$,
$= \sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c'_j}{m - m_j} + \frac{D'm + E'}{1+m^2} + C')$,
where C', D', E', c'_j are all constants.

If the domain of the subproblem is on A'_c, L intersects with a line of support on an edge e. Let m_e and p_e be the slope and intercept of e, respectively. Substituting $p = -\frac{(b_i - p_e)m^2 + (a_i - m_e b_i)m - m_e a_i - p_e}{m_e m + 1}$ into $S(L)$, we have
$S(L) = \sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c'_j m + d'_j}{(m - m_j)(1 + m_e m)}$
$+ \frac{C'm^2 + D'm + E'}{(1+m^2)(1+m_e m)} + C'')$,
where c'_j, d'_j, C', D', C'' are constants.

2.3 A Prune-and-Search approximation algorithm for solving P_1

We present a prune-and-search approximation algorithm combined with the subdivision approach. The general outline is as follows:
(1) Find upper and lower bounds for each subproblem of interest.
(2) Maintain a priority queue of subproblems based on their lower bounds.
(3) Calculate S^{min}, which is the minimum of upper bounds of all subproblems in the queue.
(4) For each subproblem I_i in the queue, if the lower bound of I_i is greater than S^{min}, prune I_i.
(5) Get the first candidate I_i from the priority queue and find a coarse approximation by placing Steiner points and sampling.
(6) A solution L is accepted, if $S(L) \le (1+\epsilon)S_i^{min}$, where ϵ is a parameter defining the quality of the approximation and S_i^{min} is the lower bound of I_i.
(7) If no acceptable solution has been found, we further divide I_i into a number of smaller subproblems.
(8) Find the upper bound and lower bound of all new subproblems and add them back to the priority queue. Go to step (3).

Next, we give more details of the prune-and-search algorithm. Given a subproblem I_i, all line segments in the feasible domain D_i intersect with the same set of regions of R. Let the set of regions intersected by segments in D_i be $\{R_{j_1}, R_{j_2}, \ldots, R_{j_k}\}$. We can set the lower bound of I_i, S_i^{min}, as $\sum_{q=1}^{k} w_{j_q} * min_{L \in D_i}(L \cap R_{j_q})$. The value of $min_{L \in D_i}(L \cap R_{j_q})$ can be computed in $O(1)$ time. The upper bound of I_i could be any sample value of the objective function in the given domain.

3 P_2: minimize the sum of weighted length of orthogonal links and L

In this section, we address the second optimization problem P_2, which asks to minimize the sum of weighted length of orthogonal links and $S(L)$, i.e. $C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$. Again, we solve this optimization problem by (1) partitioning the problem in the dual space, (2) finding the objective function on the boundary of the partition and (3) applying the prune-and-search algorithm to approximate the solution.

The problem of minimizing the sum of weighted length of orthogonal links is solved in [3]. In [3], the dual space is partitioned by introducing three sets of curves or lines. The first set consists of dual lines of points in P. Let A_p denote the arrangement of the l dual lines. Next, all lines intersecting with some orthogonal link at some edge are transformed to the dual space. This yields a set of $O(ln)$ curves. Let A_c denote the arrangement of this set of curves. The last set of lines introduced to the dual space is $\{x^* = -1/m_{ij} : i = 1, 2, \dots, l \textit{ and } j = 1, 2, \dots, n\}$, where m_{ij} is the slope of the line passing through point $s_i \in P$ and vertex $v_j \in R$. Let A_r denote the arrangement of this set of lines. The optimization problem is reduced to a number of 1-variable subproblems. The domain for each subproblem is a piece of arc or segment on A_p or A_c. The objective function for each subproblem can be expressed in the form

$$\sum_{i=1}^{l} S(L_i) = \sqrt{1+m_o^2}(\sum_{j=1}^{M} \frac{c_j}{m_o - m_j} + \sum_{i=1}^{l} \frac{b_i m_o + a_i}{m_o^2+1} + C),$$

for some constants a_i, b_i, c_i, C, where m_o is the slope of orthogonal links. Substituting $m_o = -1/m$,

$$\sum_{i=1}^{l} S(L_i) = \sqrt{1+m^2}(\sum_{j=1}^{M} \frac{-c_j}{m_j m+1} + \sum_{i=1}^{l} \frac{a_i m - b_i}{m^2+1} + \frac{C}{m}).$$

Notice that $A'_c \in A_c$, since each line of support supports an orthogonal link from a point in P to L. We can show that $A_p \cup A_c \cup A_r \cup A_v \cup A'_r$ partitions the the dual space such that each cell corresponds to all segments in the primal space which have the same functional expression for the objective function i.e. $C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$. The partition can be computed in $O(ln\log(lh)+k)$ time and $O(lh+k)$ space, where k is the number cells, which is $O(l^2n^2)$ in worst case, using the algorithm in [1].

Lemma 4 *The optimal solution for P_2 can be found at the non-vertical boundary of the partition $A_p \cup A_c \cup A_r \cup A_v \cup A'_r$.*

Lemma 5 *P_2 can be divided into a number of subproblems ($O(l^2n^2)$ in worst case) and the domain for each subproblem is a piece of non-vertical boundary of the partition $A_p \cup A_c \cup A_r \cup A_v \cup A'_r$, i.e. an arc or a segment on A_p, A_c, or A_v.*

Next, we derive the overall expression for $C(L)$. If the domain of the subproblem is on A_p or A_v, i.e. L passes through a point $s_i \in P$ or a vertex in R, we have
$C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$
$= \sqrt{1+m^2}(\sum_{j=1}^{M} \frac{-c_j}{m_j m+1} + \sum_{i=1}^{l} \frac{a_i m-b_i}{m^2+1} + \frac{C}{m}) +$
$\sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c'_j}{m-m'_j} + \frac{D'm+E'}{1+m^2} + C')$
$= \sqrt{1+m^2}(\sum_{j=1}^{M}(\frac{c''_j}{m_j m+1} + \frac{c'_j}{m-m_j}) + \frac{D''m+E''}{1+m^2} + \frac{C}{m} +$
$C')$,
where c'_j, c''_j, C, C', D'' and E'' are constants.

If the domain of the subproblem is on A_c, i.e. L intersects with an orthogonal link or a line of support on an edge, we have,
$C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$
$= \sqrt{1+m^2}(\sum_{j=1}^{M} \frac{-c_j}{m_j m+1} + \sum_{i=1}^{l} \frac{a_i m-b_i}{m^2+1} + \frac{C}{m}) +$
$\sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c'_j m+d'_j}{(m-m'_j)(1+m_e m)} + \frac{C'm^2+D'm+E'}{(1+m^2)(1+m_e m)} +$
$C''')$
$= \sqrt{1+m^2}(\sum_{j=1}^{M}(\frac{c''_j}{m_j m+1} + \frac{c'_j m+d'_j}{(m-m_j)(1+m_e m)}) +$
$\frac{C''m^2+D''m+E''}{(1+m^2)(1+m_e m)} + \frac{C}{m} + C''')$,
where c'_j, c''_j, d'_j, C'', D'', E'' and C''' are constants.

Once the objective functions are derived for all subproblems, the optimal solution can be obtained by applying the prone-and-search algorithm described in the previous section.

References

[1] N.M. Amato, M.T. Goodrich, and E.A. Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. *In Proc. 11th Annual CAM-SIAM Symposium on Discrete Algorithms*, pp. 705-706, 2000.

[2] D.Z. Chen, O. Daescu, X. Hu, X. Wu and J. Xu. Determining an optimal penetration among weighted regions in two and three dimensions. *Journal of Combinatorial Optimization*, 5(1):59-79, 2001.

[3] Y. Cheung and O. Daescu. Line Facility Location in Weighted Regions. *In Proc. 4th International Conference on Algorithmic Aspects in Information and Management*, pp. 109-119, 2008

[4] O. Daescu and J. Palmer. Minimum Separation in Weighted Subdivisions. *International Journal of Computational Geometry and Applications* to appear.

[5] H. Imai, K. Kato and P. Yamamoto. A linear-time algorithm for linear L_1 approximation of points. *Algorithmica*, 4(1):77-96, 1989.

[6] M. X. Goemans, M. Skutella. Cooperative facility location games. *In Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pp. 76-85, 2000.

[7] N. Megiddo and A. Tamir. Finding Least-Distance Lines. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):207-211, 1983.

[8] G.T. Toussaint. Solving geometric problems with the 'rotating calipers'. *In Proc. 2nd IEEE Mediterranean Electrotechnical Conference (MELECON '83)*, pp. A10.02/1-4, 1983.

Improvement of the Method for Making Quad Meshes through Temperature Contours

Minori Okabe* Shinji Imahori* Kokichi Sugihara*

Abstract

There are several methods to reconstruct quad meshes from triangular meshes. In this article, we improve the method using temperature contours. We discuss how to avoid boundary conditions being ill posed and assure injectivity of the parameterization by controlling the weights on edges. The proposed method will be one step toward a fully automatic system, because the result of automatic partition of the surface into patches can be used.

1 Introduction

Triangular meshes are often used for representing surfaces of objects in computer graphics and simulation. Whereas quad meshes have the advantage of triangular meshes in many applications such as texture mapping in computer graphics, FEM in simulation and NURBS in modeling.

Constructing quad meshes directly from point clouds is difficult because we do not have efficient tools such as Voronoi diagrams in case of creating triangular meshes. Thus most of methods reconstruct triangular meshes into quad meshes.

One major trend is following some vector fields on the mesh. Principal directions are often chosen as the vector fields. One approach with vector fields is tracing them by numeric integration [1]. Another approach is defining functions on the mesh whose isolines follow the vector fields [7, 6]. This approach needs solving non-linear optimization instead of numeric integration. Another trend is defining functions on the given triangular mesh by parameterization [3, 8], which only needs linear computation and gives pure quad meshes.

We propose a new method to generate quad meshes, which is based on and improves the method by tong et al. [8]. In this method, quad meshes are obtained through temperature contouring. It is simple and efficient in calculation, but not enough from a practical viewpoint. We discuss the feasibility of an optimization problem appearing in the algorithm and assure injectivity of the parameterization by controlling the weights on edges.

*Graduate School of Information Science and Technology, The University of Tokyo, {minori_okabe, imahori, sugihara} @mist.i.u-tokyo.ac.jp

2 Outline of the proposed method

We describe the outline of our algorithm, which is similar to that of tong's method [8].

To achieve quad meshes, we give each vertex of the given triangular mesh u and v values, and draw integer contour lines from them. Assigning two values is equivalent to planar parameterization. Because the given mesh is not homeomorphic to a disk in general, we have to divide it into some patches homeomorphic to disks. However, if patches are mapped into the plane independently, the continuity of contours is not assured, which is a requirement for quad meshes. For this purpose, some conditions of "jump" and "rotation" on patch boundaries are necessary. In case that the function value of a continuous contour line changes across a patch boundary, it should jump by a certain integer value. Moreover, in case that a contour line changes its function and direction, that is, it switches from u to v, u to $-u$ and so on, the rotating angle of the u-v coordinates should be equal to an integral multiple of $\frac{\pi}{2}$.

Now, the outline of our method is as follows:

1. Divide the mesh into patches.
2. Decide the connectivity conditions between patches — jump values and rotating angles.
3. Solve a linear system to obtain u and v values for each vertex and draw contours.

Hereafter, we call the mesh derived from the division of the original mesh into patches *meta-mesh*, and vertices, edges and facets of the meta-mesh *meta-vertices*, *meta-edges* and *meta-facets* respectively. A vertex meeting three or more patches corresponds to a meta-vertex. To make pure quad meshes, whose facets have just four vertices, u and v values of meta-vertices should be integer in each adjacent patches. We note that meta-vertices are the candidates of singular vertices of the resulting quad meshes, where singular vertices mean vertices adjacent to other than four facets.

3 Details of each procedure

3.1 Division into patches

The first procedure is to divide the original triangular mesh into patches homeomorphic to disks. As this division may affect the resulting quad meshes, it should reflect some geometric features of the mesh. Here we use VSA method [2], in which each divided region is designed to approximate the plane through alternate update of the approximate planes and the patch division.

In VSA method, the patches tend to meet where the curvature is relatively large. This is a desirable property for quad meshes. Moreover, we can roughly control the number of singular vertices by adjusting the number of patches.

It is to be noted that some inputs from VSA method are rejected since some patches are not homeomorphic to disks. It is because VSA method guarantee each patch to be connected, but not to be homeomorphic to a disk. To resolve this problem, patches should be cut into subpatches homeomorphic to disks.

3.2 Solving a linear system

We explain the last procedure before the second one for understanding. Similar to the ordinary planar parameterization, u and v values of a vertex is set to the weighted average of u and v values of the neighboring vertices:

$$\sum_{j\in n_i} w_{ij} \begin{pmatrix} u_i - u_j \\ v_i - v_j \end{pmatrix} = 0,$$

where n_i is the set of the neighboring vertices of vertex i. Here, we use the mean value coordinates [5] as weights w_{ij} (see Section 3.4).

But for vertices on boundaries between patches, this formula is not sufficient because we admit jumps on u and v values and rotation of the u-v coordinates. Let vertex i be on the patch named '$-$', and patch '$+$' be one of adjacent patches of patch '$-$'. n_i^+ (resp., n_i^-) means the partial set of n_i on patch '$+$' (resp., '$-$').

For example, if u values differ by p_1 from one patch to another and v by p_2, the condition is as follows:

$$\sum_{j\in n_i^-} w_{ij} \begin{pmatrix} u_i - u_j \\ v_i - v_j \end{pmatrix} + \sum_{j\in n_i^+} w_{ij} \begin{pmatrix} u_i - (u_j + p_1) \\ v_i - (v_j + p_2) \end{pmatrix} = 0.$$

In case of rotation, when the contours of u switched into contours of v and v to $-u$ and the jump values are r_1 and r_2, the condition can be written down as follows:

$$\sum_{j\in n_i^-} w_{ij} \begin{pmatrix} u_i - u_j \\ v_i - v_j \end{pmatrix} + \sum_{j\in n_i^+} w_{ij} \begin{pmatrix} u_i + (v_j - r_1) \\ v_i - (u_j + r_2) \end{pmatrix} = 0.$$

Conditions for other rotations (i.e., 180° and 270°) can be written down similarly.

u and v values of meta-vertices are given in advance by the second procedure. This is corresponding to boundary conditions of planar parameterization. Injectivity of this mapping is referred later in Section 3.4.

3.3 Deciding parameters

We decide the jump values and rotating angles for meta-edges and u and v values of meta-vertices. Once the region on the u-v plane to which each patch is mapped is assigned, the connectivity of patches is automatically derived.

When two meta-vertices are adjacent on the meta-mesh, they should be connected by a contour line. That is, each meta-vertex is on a lattice point of the u-v plane and each meta-facet is mapped into a polygon with right angles on the u-v plane. By this condition, determining the shape results in simple calculations. In addition to the method in [8], we restrict the shape to a rectangle in order to avoid self intersection and assure injectivity. Thus what should be done is to decide the angles of meta-vertices on each patch and the lengths of meta-edges.

The angles for each patch is set to minimize the difference of the length of facing edges. Here, we use the shortest path length on the original mesh instead of the actual path length of the boundaries in order to make quad meshes equilateral.

Next we determine the length of each meta-edge. For each facet, the sums of the lengths of facing meta-edges must be equal. This can be represented as a linear condition $A\boldsymbol{x} = 0$, where $\boldsymbol{x}$ is the variants vector corresponding to the lengths of meta-edges. We consider minimizing the difference from the shortest path length of corresponding two vertices on original mesh represented by $\boldsymbol{c}$:

$$\begin{aligned} &\min \ \|\boldsymbol{x} - \boldsymbol{c}\|, \\ &\text{s.t. } A\boldsymbol{x} = 0, \\ &\qquad \boldsymbol{x} \in \mathbb{Z}_+^n, \end{aligned}$$

where n is the number of meta-edges and $\mathbb{Z}_+$ is the set of positive intergers. By using 1-norm, this problem can be represented as a linear programming problem with integer conditions:

$$\begin{aligned} &\min \ t_1 + t_2 + \cdots + t_n, \\ &\text{s.t. } A\boldsymbol{x} = 0, \\ &\qquad -t_i \le c_i - x_i \le t_i \ (i \in \{1, \dots, n\}), \qquad (*) \\ &\qquad \boldsymbol{x} \in \mathbb{Z}_+^n, \\ &\qquad t_i \ge 0 \ (i \in \{1, \dots, n\}). \end{aligned}$$

The feasibility of this problem is discussed later in Section 5.

If the shape of each patch is obtained through the angles of meta-vertices and the lengths of meta-edges, u and v values of meta-vertices are specified by fixing arbitrarily u and v values for one meta-vertex and the direction of one meta-edge. After doing this for each patch, jump values and rotating angles between patches are automatically obtained.

3.4 Guarantee for injectivity

Since injectivity is not guaranteed, triangles may flip, which results in no pure quad meshes. In addition, the triangles around singular vertices have a tendency to flip and this hurts the quality of the resulting quad mesh.

In ordinary planar parameterization, a theorem in [5] is known to guarantee the injectivity. From this theorem, a mapping is guaranteed to be injective if (1) a triangular mesh homeomorphic to a disk is 3-connected, (2) the weights on edges have positive values, and (3) the boundary is mapped into a convex polygon. To apply this theorem to each patch, we examine the three conditions — 3-connectedness, positivity and convexity.

3-connectedness: For a mesh homeomorphic to a disk, the graph derived from it is 3-connected if there exists no edge which is not on any patch boundary but whose adjacent vertices are both on patch boundaries. When such an edge exists, we insert a vertex on the edge and divide the two adjacent triangles accordingly.

Moreover we do the following procedure in advance. We check whether edges whose two adjacent vertices are on the same patch boundary exist or not. If there exist, we convert the boundary to use these edges. This modification does not change the topology of the meta-mesh. It has some smoothing effect on the resulting quad mesh because the boundaries of patches may be the edges of the resulting quad mesh.

Positivity: The mean value coordinates [4] are chosen as weights because of its positivity.

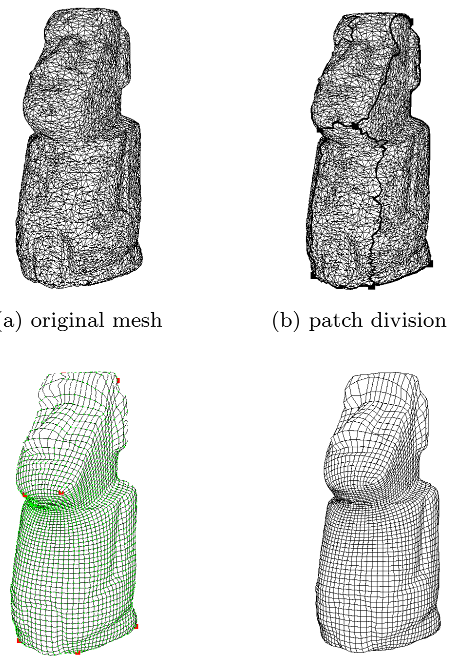
Convexity: When boundaries are mapped into convex polygons, each patch is mapped into a rectangle because convex polygons with right angles are surely rectangles. Weights of edges on patch boundaries are set to extremely large values. Then the vertices on patch boundaries are placed on the line between two meta-vertices. In result, the vertices on the boundaries are positioned into a convex shape.

Under these conditions injectivity of the mapping is guaranteed. But this hurts the quality of the resulting quad mesh because the seams become visible. To avoid this, we decrease the weights of the edges on the patch boundaries toward the original weights unless there exist flipped triangles. If triangles are flipped by the decreasing to the original weight, we use a binary search to find smaller weights under the condition that any triangle is not flipped.

4 Experimental results

In this section we show some experimental results. We first show results along the overall process in Figure 1. The original triangular mesh has about 7000 vertices (Figure 1(a)). It is divided into 7 patches as shown in Figure 1(b). Figure 1(c) shows the contour lines of u and v values and Figure 1(d) shows the resulting quad mesh.

Square points in Figure 1 mean meta-vertices. The whole calculation needs a few minutes on an ordinary personal computer.


(a) original mesh (b) patch division

(c) contours (d) quad mesh

Figure 1: Process from a triangular mesh to a quad mesh

Next we show the effect of controlling the weights of edges on patch boundaries. In Figure 2(a), the mean value coordinates are used as weights. The meta-vertex in the middle has only one contour because some of triangles around this are flipped. On the other hand, the weights of the edges on the patch boundaries are added by a sufficiently large value in Figure 2(b). In this case no triangle is flipped and all the meta-vertices have the appropriate numbers of contours.

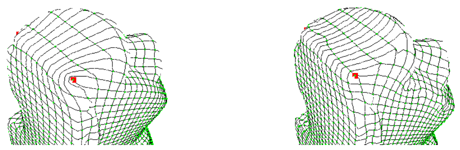

(a) (b)

Figure 2: The comparison of contours between original weights and increased weights

Finally, we show how increased weights damage the quality of the resulting quad meshes and it is refined by adjusting weights. The weights of the edges on the patch boundaries are added by a sufficiently large value in Figure 3(a). On the other hand, these weights are cut down as long as there exists no flipping triangle in Figure 3(b). It is found that smoothness increases as a result of cutting the weights down.

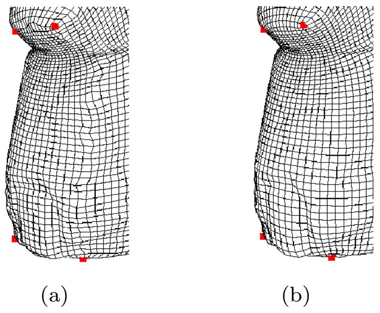


Figure 3: The comparison of quad meshes between increased weights and adjusted weights

5 Discussion on the feasibility

In this section we discuss on the feasibility of the optimization problem(∗) appearing when deciding the length of meta-edges (Section 3.3). This problem may not have feasible solutions. See Figure 4 as an example.

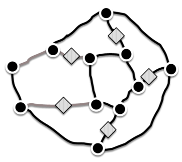


Figure 4: An example of edges with 0 length — if gray boundaries are set to be facing on the u-v plane, the length of the upper left boundary is inevitably equal to 0.

One solution is to divide patches until each patch has four meta-vertices because there is at least one feasible solution (i.e., all meta-edges have the same length). If a patch has even meta-vertices, it is possible by inserting meta-edges. In contrast, in case of a patch with odd meta-vertices, it is impossible without inserting meta-vertices. In this case, by inserting meta-vertices on meta-edges derived from a path of patches from a patch with odd meta-vertices to other patch with odd meta-vertices, patches with odd meta-vertices can be converted to have even meta-vertices without changing the parity of other patches (see Figure 5). It is possible because the number of patches with odd meta-vertices is surely even.

6 Conclusion

In this article, we improved the method for constructing quad meshes using temperature contours from a

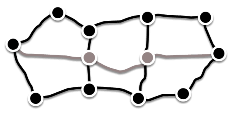


Figure 5: An example of inserting meta-vertices — each patch at both ends has an odd number of meta-vertices. By inserting two gray meta-vertices, all the patches have even meta-vertices and are divided so as to have four meta-vertices.

practical viewpoint. We discussed on the feasibility of the optimization problem and proposed an approach of splitting meta-facets. Moreover we assured injectivity of the mapping by adjusting the weights of edges on patch boundaries. Increased weights assure injectivity but also damage the quality of the resulting quad meshes. In order to adjust weights, we used a bisection algorithm which requires iterative solving of linear systems. Working out more efficient methods for weight adjustment is an issue in the future.

Acknowledgments

We would like to thank CGAL library. The model in the figures is courtesy of AIM@SHAPE. This work is supported by the Grant-in-Aid for Scientific Research(b) (No. 20360044) and for Exploratory Research (No. 19650003) of the Japanese Society for Promotion of Science.

References

[1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, 2003.

[2] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, 2004.

[3] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. Spectral surface quadrangulation. *ACM Trans. Graph.*, 25(3):1057–1066, 2006.

[4] M. S. Floater. Mean value coordinate. *Comput. Aided Geom. Des.*, 20(1):19–27, 2003.

[5] M. S. Floater. One-to-one piecewise linear mappings over triangulations. *Math. Comput.*, 72(242):685–696, 2003.

[6] F. Kälberer, M. Nieser, and K. Polthier. Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum*, 26(3):375–384, Sept. 2007.

[7] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez. Periodic global parameterization. *ACM Trans. Graph.*, 25(4):1460–1485, 2006.

[8] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proc. SGP* (2006), 201–210, Switzerland, 2006. Eurographics Association.

From Mesh Parameterization to Geodesic Distance Estimation

Joachim Giard * †　　　Benoît Macq *

Abstract

Computing geodesic distances on a mesh can be a time consuming, though redundant task. In this paper, we propose a method to reuse the information obtained during the computation of geodesic distances from some reference vertices to estimate distances between other vertices. First, the mesh is parameterized, unfolding it in a plane around each reference vertex. Then, distances between vertices are computed using simple operations in these parameterized planes. It quickly leads to a good approximation of geodesic distances on the mesh.

Keywords: Geodesic distance, parameterization, Unfolding, 3D Mesh.

1 Introduction

Computing geodesic distances on 3D meshes, i.e., distances along the surface, can be really time consuming, especially in applications requiring *"all sources to all targets"* computations. Mitchell et al. [6] introduced an exact *"single source to all targets"* algorithm running in $O(n^2 \log n)$. This algorithm is close from the Dijkstra algorithm [3] finding shortest paths in graphs. In 1998, Kimmel and Sethian [4] presented the fast marching algorithm running in $O(n \log n)$.
In methods exposed here above, *"single source to single target"* or *"single source to all targets"* problems are treated, but often *"all sources to all targets"* problems have to be considered. It is the case, for instance, when geodesic distance is used as a basis for discrete differential-geometry operators [5] such as curvature estimators, integrators or surface field filters. Using *"single source to all targets"* algorithms to compute such distances, without taking into account redundant information, multiplies the complexity and the effective computation time by n.
In this paper, we propose a method to quickly estimate geodesic distance using parameterization and neighborhood information. The general idea of the method is to parameterize the mesh by unfolding it around references vertices and to compute distances in the plane. For source vertices which are not references, the parameterization of the closest reference vertex is used as if the mesh has been unfolded around it.
Other methods were proposed in the literature to reduce the complexity of such calculations. For instance, the number of vertices on which the distance is computed can be reduced [1]. In the method described here, all the vertices are conserved, but the dimension of the calculation space is reduced, keeping a good resolution and then a good precision around interest points.

2 Method

On a mesh, if two vertices are geodesically close, they have a similar geodesic distances map. In particular, if a vertex p_c lies on the geodesic between two other vertices, p_a and p_b, the geodesic distance from p_c to p_b, $D_c(b)$, can directly be deducted from $D_a(b)$ and $D_a(c)$, both computed from the same vertex, p_a. This reasoning can be extended to vertices lying on different geodesics, bringing the problem into the plane instead of along a line and resolving a triangle. Then, it is possible to reuse information extracted during the computation of geodesics from a particular vertex for the approximation of geodesics from other vertices. By the way, in practise, it would be really space-consuming to keep all the geodesics in memory. So, in the proposed method, a few reference vertices are selected and a geodesic distances map is computed for each of these vertices. This distance map is computed in a plane containing a parameterization of the shape after an unfolding. The method is divided into a preparation phase, during which a parameterization of the mesh is computed around some reference vertices, and a estimation phase, during which the parameterizations are used to estimate distance maps for concrete applications.

2.1 Preparation Phase

During the preparation phase, some reference vertices are selected and the mesh is parameterized around each of them. A parameterization is a one-to-one mapping from a surface to another one. Many ways to map vertices of a 3D mesh on other surfaces such as spheres or planes are found in the literature [7]. One of the major application of parameterizations is

*Communications and Teledetection Lab., Université catholique de Louvain, Belgium.
†Communication author: joachim.giard@uclouvain.be.
J.G. is funded by Nanotic/Tsarine, project n° 516248 of the Région Wallonne of Belgium.

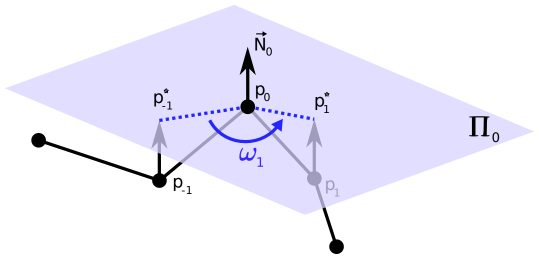


Figure 1: Unfolding of edges in the tangent plane Π_0. p_0 is the current active vertex, p_{-1} is its parent in the unfolded tree and p_1 is a non-visited vertex. p_{-1} and p_1 are projected in the tangent plane along the normal vector $\vec{N}_0$. Given the length of $[p_{-1}p_0]$, the length of $[p_0p_1]$, the projected angle ω_1 and the parameterizations of p_{-1} and p_0, it becomes possible to compute the parameterization of p_1.

texture mapping [8]. Generally, geodesic distance are needed to produce the parameterization. For this method, a new parameterization is used because it is needed to estimate the geodesic distances and not the opposite. The parameterization is designed to be computationally efficient and to guarantee that close vertices have similar parameterizations. This technique is a mix of shape unfolding [2] and of the fast marching algorithm for geodesic distances computation [4]. It maps vertices of the mesh to a plane parameterized with a distance and an angle. Let p_r be the selected reference vertex and $D_r(i)$, the geodesic distance from p_r to p_i. At the first step, $D_r(r) = 0$ and $D_r(i) = \infty$ if $p_r \neq p_i$. Then, the active vertex p_0 is chosen as the non-visited vertex with the smallest distance. At the first step $p_0 = p_r$. At each step, vertices connected to p_0 are unfolded by rotation of edges to bring them in the tangent plane of p_0 (see Figure 1). Let p_1 be one of these connected vertices and p_{-1} the parent of p_0 in the unfolded tree. $p^\star$ is the projection of p in the unfolding plane. The angle $\omega_1 = \widehat{p^\star_{-1}p_0p^\star_1}$ allows to compute the parameterization of p_1, and to update the distance from p_r. At the end of the step, $D_r(1)$ and $\alpha_r(1)$, the polar parameters of p_1 are updated, p_0 is recorded to be the parent of p_1 and is labelled as visited. Then, the new p_0 is selected and so on until all vertices are visited. A simple polyhedron and an unfolded version are depicted in Figure 2. The unfolding by successive projections on the tangent plane does not conserve neither the angles nor all the connectivity, but has some advantages for this method. Geodesics have to be parameterized as straight lines, so the distance between two vertices can be computed using a simple operation. Conserving angles does not parameterize geodesics as approximated straight lines, what leads to crossing risks and obstruct the calculation of distances by simple operations.

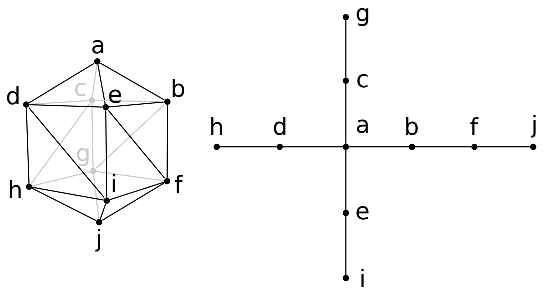


Figure 2: A simple polyhedron (left) and its unfolded version (right) around the vertex a.

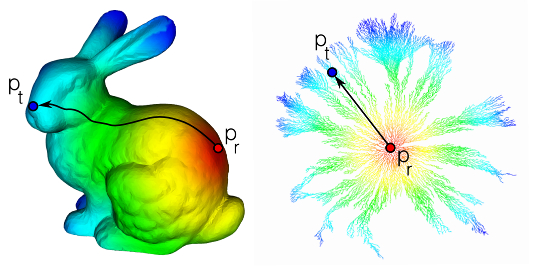


Figure 3: A distance map centered in the reference vertex p_r, and a geodesic between p_r and a target vertex p_t. The distance map is first computed on the unfolded version (right) and is mapped on the 3D shape (left). Distances on the unfolded version are computed as straight segment lengths in the plane.

An example of a distance map computed on a mesh based on its unfolding is depicted in Figure 3[1]. This parameterization is performed for all the references vertices and the parameters are kept in memory. A new reference vertex is selected as the vertex of the mesh being the geodesically furthest one from all other reference vertices.

2.2 Estimation Phase

During the estimation phase, parameterizations computed in the preparation phase are used to estimate geodesic distances from any vertex of a mesh. Let p_s and p_t be respectively the source and the target vertex in a practical application. Let p_r be the geodesically closest reference vertex from p_s. The parameters of all the vertices based on the unfolding around p_r are stored in the vectors D_r (distance) and α_r (angle). The approximation of the geodesic distance between p_s and p_t, $\tilde{D}_s(t)$, is computed using the law of cosines:

$$\gamma^r_s(t) = \alpha_r(t) - \alpha_r(s),$$

$$\tilde{D}_s(t) = \sqrt{D^2_r(s) + D^2_r(t) - 2D^2_r(s)D^2_r(t)\cos\left(\gamma^r_s(t)\right)}.$$

[1]Stanford Bunny, courtesy of Greg Turk and Marc Levoy.

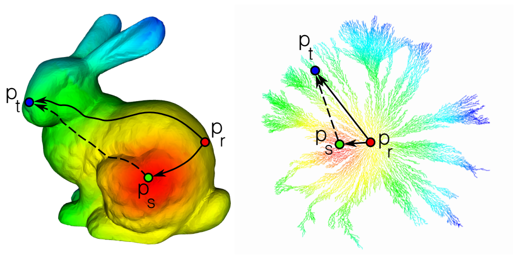


Figure 4: Estimation of the distance map centered in a source vertex p_s and a geodesic estimation between p_s and a target vertex p_t. The geodesic distance between p_s and p_t is estimated in the unfolding plane around p_r (right) based on the lengths of $\overrightarrow{p_r p_s}$ and $\overrightarrow{p_r p_t}$ and the angle between both. The estimated distance map is mapped on the 3D shape (left).

An approximation example is shown in Figure 4. This application is based on the parameterization around p_r, depicted in the Figure 3.

3 Results

The aim of this method is to provide quickly an estimation of geodesic distances reusing information computed for other source vertices. To show the time advantages, a complexity analysis and execution times for typical tasks are presented in this section. Some estimation error measurements are also presented in this section.

3.1 Complexity Analysis

The time complexity for a mesh containing n vertices for a *"single source to single target"* estimation is $O(mn \log n) + O(1)$, where m is the number of reference vertices. The time complexity for the parameterization around a vertex is $O(n \log n)$ and this operation is repeated m times during the preparation phase. So the fixed cost of the algorithm is $O(mn \log n)$. The variable cost depends directly on the number of sources and targets. It is the cost of the estimation phase which only contains simple operations. The *"all sources to all targets"* algorithm has a complexity of $O(mn \log n) + O(n^2)$ instead of $O(n^2 \log n)$ without estimation (in this case, $m = n$, and no estimation phase is needed).

3.2 Execution Times

Execution times[2] for *"all sources to all targets"* tasks are presented in Table 1. The algorithm was applied on three meshes: the Sphere (1522 vertices), the Bunny (14007 vertices) and the Horse (48485 vertices). Execution time for the parameterization step around the n vertices of each mesh appears in the left column. Execution times for the estimation of the distances from all the vertices, based on a reference parameterization, appear in the right column. Since both *"all targets"* processes depends linearly on the number of sources, the total execution time for p source vertices based on m reference vertices is the left column time weighted by $\frac{m}{n}$ added to the right column time weighted by $\frac{p}{n}$. For instance, the execution time for 1000 sources based on 100 reference vertices on the Bunny is $\frac{100}{14007}7250 + \frac{1000}{14007}32 \simeq 54s$.

Mesh Name	Parameterization	Estimation
Sphere	$43s$	$0s$
Bunny	$7250s$	$32s$
Horse	$194000s$	$323s$

Table 1: Execution times of *"all sources to all targets"* tasks for the parameterization alone (left) and for the estimation alone (right). The test were performed on three meshes (Sphere, Bunny and Horse). In the method presented here, the parameterization does not need to be *"all sources to all targets"* but *"some reference sources to all targets"*, making the total parameterization time shorter.

3.3 Estimation Errors

For the geodesic distance based on the parameterization, some minor errors can appear. First, the parameterization make the geodesics "follow" edges, whereas sometimes, going through a face would be shorter. This error becomes smaller if the size of the faces becomes smaller. Second, some crossings can appear in the unfolding, leading to discontinuities in the distance map. However, for relatively regular surfaces, these crossings are generally far from the source, where less precision is needed.
Bigger errors appear in the estimation phase. First, if the number of reference vertices is too low, two close vertices can be considered to be far from each other if they are on the border of the parameterization (overestimation). The error rate is unbounded because two close vertices can belong to different reference vertex zones, so an infinitely small distance may be estimated as a larger positive value. Second, if the surface is not smooth and if the geodesics from reference vertices follow valleys and crests, then, the geodesics from another source vertex, could cross the ripples without taking them into account (underestimation). The underestimation rate is unbounded because the number of ripples can be very high even close from the reference vertex. Mean error percentages for two meshes, the Sphere (1522 vertices) and the Head (367 vertices), depending on the proportion of reference ver-

[2]The algorithm was implemented in C++ with the Visual ToolKit. Tests were performed on an Intel Core 2 CPU @ 2.00GHz, and with 2 Gb RAM.

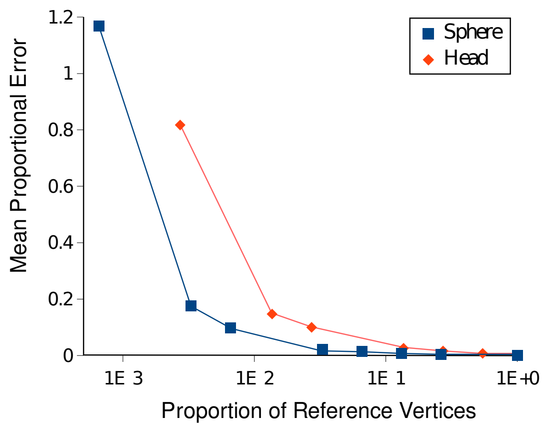


Figure 5: Error graph of the estimation phase as a function of the proportion of reference vertices. The proportional error was calculated on two 3D shapes: the Sphere (blue upright squares), which is simple and convex, and the Head (red rotated squares). The scale of the x-axis is logarithmic.

tices are depicted in Figure 5. It shows the errors coming from the estimation, so the geodesic distances computed from the parameterization were considered to be exact. The mean proportional error, ϵ, is computed as follow:

$$\epsilon = \frac{1}{\sigma^2}\sum_{i=1}^{n}\sigma_i\left(\sum_{j=1}^{n}\frac{|D_i(j)-\tilde{D}_i(j)|}{D_i(j)}\sigma_j\right),$$

where σ is the total area of the mesh and σ_j is a proportion factor depending on faces area around the vertex p_j.

4 Conclusion

In this paper, we presented a method to quickly estimate geodesics distances on meshes. This method is based on parameterizations around some reference vertices, unfolding the mesh in a plane and computing distances using simple operations in this plane. As showed in section 3, for "classical" meshes, the local errors produced by this estimation are below 5% if the reference vertices proportion is over 10%. Since the execution time of the estimation phase is negligible in comparison with the parameterization time, taking only 10% of reference vertices divides the execution time by 10.
With the current used technique, the error is not bounded. But in practise, with relatively smooth meshes, the approximation works well. The error can be theoretically bounded by using other technique to rely application vertices to reference vertices. For instance, by combining several references or by considering an angle difference instead of a distance to choose them. In future works, we will prove some intuitive ideas exposed here, and we will continue to improve the efficiency of the algorithm by choosing, for instance, a better parameterization, or by selecting reference vertices in a more efficient way.
The algorithm will also be used by members of the team in molecular docking and 3D watermarking processes.

References

[1] L. Aleksandrov, A. Maheshwari, and J. Sack. Approximation algorithms for geometric shortest path problems. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 286–295. ACM New York, NY, USA, 2000.

[2] E. Demaine and J. ORourke. A survey of folding and unfolding in computational geometry. *Combinatorial and Computational Geometry*, 2005.

[3] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[4] R. Kimmel and J. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences*, 95(15):8431–8435, 1998.

[5] M. Meyer, M. Desbrun, P. Schroder, and A. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and Mathematics*, 3:35–57, 2003.

[6] J. Mitchell, D. Mount, and C. Papadimitriou. The Discrete Geodesic Problem. *SIAM Journal on Computing*, 16:647, 1987.

[7] A. Sheffer, E. Praun, K. Rose, and I. NetLibrary. Mesh Parameterization Methods and Their Applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2):105–171, 2006.

[8] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):198–207, 2002.

Certified Meshing of RBF-based Isosurfaces

Amit Chattopadhyay[*] Simon Plantinga [†] Gert Vegter[‡]

Abstract

Radial Basis Functions are widely used in scattered data interpolation. The process consists of two steps: (i) computing an interpolating implicit function the zero set of which contains the points in the data set, followed by (ii) extraction of isocurves or isosurfaces. We focus on the second step, generalizing our earlier work on certified meshing of implicit surfaces based on interval arithmetic. It turns out that interval arithmetic, and even the usually faster affine arithmetic, are far too slow in the context of RBF-based implicit surface meshing. We present optimized strategies giving acceptable running times and better space complexity, exploiting special properties of RBF-interpolants. We present pictures and timing results confirming the improved quality of these optimized strategies.

1 Introduction

RBF-based interpolants. Radial Basis Functions provide a simple meshless method for the reconstruction of smooth geometric objects in the plane or in three-dimensional space from a finite point sample $v_1, \ldots, v_n$. The process consists of two steps: (i) computing an interpolating implicit function the zero set of which contains the sample points, followed by (ii) extraction of the isocurve or isosurface.

The Radial Basis interpolant constructed in step (i) is of the form

$$s(\underline{x}) = \sum_{k=1}^{n} w_k \, \varphi(||\underline{x} - v_k||) + p(\underline{x}), \qquad (1)$$

where $\underline{x} \in \mathbb{R}^d$, for $d = 2, 3$, such that s is zero at the sample points (*centers*) v_k. Here p is a polynomial of low degree, cf [5]. The Radial Basis Function (RBF) φ is a univariate function. Some popular RBFs are $\varphi(r) = r^3$ (triharmonic spline in 3D), $\varphi(r) = r^2 \log r$ (thin plate spline in 2D), $\varphi(r) = \sqrt{r^2 + c^2}$ (multiquadric), $\varphi(r) = \exp(-r^2)$ (Gaussian).

The second step, namely isosurface extraction, is our main focus. In [9] we use *interval arithmetic (IA)* to extract regular level sets of a general smooth (C^1) implicit function. More precisely, the algorithm computes a piecewise linear surface which is close (isotopic) to the actual zero set, and is guaranteed to have the same topology. It is akin to the Marching Cubes algorithm in the sense that it analyzes the topology of the isosurface on boxes in the plane or in space. If it cannot decide that the topology is correct, it subdivides the box. However, interval arithmetic converges very slowly for implicit functions like (1), i.e., sums consisting of a large number of terms.

Our contribution. Our early experiments show that even the straightforward use of *affine arithmetic (AA)* [4], a fine tuned version of IA, does not improve running times sufficiently. Therefore, we developed an improved strategy using *linear* upper and lower bounds, exploiting the fact that each term in the sum is of the same form. This strategy works for certain RBFs, and leads to spectacular improvement of the running time, since far less subdivisions of boxes are needed before the algorithm can decide that the topology is correct. Since such linear bounds are not easy to obtain for all types of RBFs we also developed a more general method based on *quadratic* bounding functions, which works for commonly used RBFs. Finally, we give pictures and performance results confirming the improved quality of the optimized strategy in terms of time and space complexity.

Related Work. Current methods for meshing RBF-based implicit surfaces do not come with topological guarantees, since they are usually based on the marching cubes algorithm [7]. Methods for certified meshing of implicit surfaces are presented in [3, 10]. In [9] interval arithmetic is used to extract certified meshing of implicit surfaces. For an overview of interval arithmetic methods and their optimizations we refer to [8]. Affine arithmetic is discussed in [4].

2 Preliminaries

Interval Arithmetic (IA). Interval arithmetic is used to prevent rounding errors in finite precision computations. A *range function* $\Box F$ for a function $F : \mathbb{R}^m \to \mathbb{R}^n$ computes for each m-dimensional interval I (i.e., an m-box) an n-dimensional interval $\Box F(I)$, such that $F(I) \subset \Box F(I)$. A range function is said to be *convergent* if the diameter of the output interval converges to 0 when the diameter of the input interval shrinks to 0. Convergent range functions exist for the basic operators and functions, so all range functions are assumed to be convergent.

[*]Corresponding author. University of Groningen, The Netherlands; email: `A.Chattopadhyay@rug.nl`
[†]Email: `S.Plantinga@rug.nl`
[‡]Email: `G.Vegter@rug.nl`

Certified Meshing Algorithm. The certified meshing algorithm [9] subdivides the domain of an implicit function until it can approximate the zero set of the function in each box with a topologically correct piecewise linear surface. The algorithm takes an implicit function F and a box B as input, and computes a piecewise linear approximation of $F^{-1}(0) \cap B$, assuming that the zero set $F^{-1}(0)$ of F contains no singular points of F inside B. It uses range functions for F and its gradient ∇F.

Algorithm: ApproximateCurve(F, B)

```
1. Initialize quadtree $T$ to $B$;
2. Subdivide $T$ until for all leaves $I$:
        $0 \notin \Box F(I) \vee \langle \Box\nabla F(I), \Box\nabla F(I)\rangle > 0$;
3. Mesh$(T)$.
```

Here, Mesh(T) approximates the zero level set inside the box T by a linear function. The first clause in line 2 discards cells I for which $0 \notin \Box F(I)$, i.e., boxes which are guaranteed not to contain part of the zero set of F. The second clause implies that $\langle \nabla F(x), \nabla F(y)\rangle > 0$, for all $x, y \in I$, so the direction of the gradient (and, therefore, of the curve) does not change by more than $\pi/2$ over this box. This implies that the zero set of F is *parametrizable* (i.e., can be written as a function of x or y), which is the key property in the proof of topological correctness of the output. We refer to [9] for details.

3 Range functions for RBFs

Unfortunately, for RBF-based implicit functions s of the form (1) an IA-based implementation of algorithm ApproximateCurve(s, I) has unacceptable running times. Our goal is to improve the performance considerably by optimizing the range intervals $\Box s(I)$ and $\Box\nabla s(I)$ for such RBF-interpolants s on a box I. We restrict our discussion to the two-dimensional case, althought our approach works in any dimension.

Computing $\Box s(I)$. Our optimization strategy determines a lower bound $l_k(\underline{x})$ and an upper bound $u_k(\underline{x})$ for $w_k\varphi(||\underline{x} - v_k||)$ on the box I, such that the minimal value $L(I)$ of $l(\underline{x}) := \sum_k l_k(\underline{x}) + p(\underline{x})$ on I and the maximum value $U(I)$ of $u(\underline{x}) := \sum_k u_k(\underline{x}) + p(\underline{x})$ on I are easy to compute. Moreover, taking $\Box s(I) = [L(I), U(I)]$ should yield a much better range interval than AI, or even AA.

Our approach is based on the observation that the summand $w_k\varphi(||\underline{x} - v_k||)$ is radially symmetric with respect to the center v_k. We will find *quadratic* upper and lower bounds for the *univariate* function $w_k\,\varphi(r)$ for r ranging over the smallest interval $J_k = [r_1, r_2]$ for which $r_1^2 \le ||\underline{x} - v_k||^2 \le r_2^2$, for all $\underline{x} \in I$. See Figure 1. More precisely, the univariate upper bound

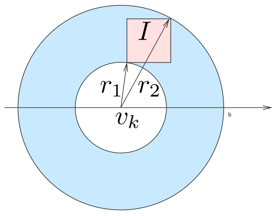


Figure 1: Near and far point of a square interval I. If the center v_k lies inside the box I, then $r_1 = 0$.

of $w_k\varphi(r)$ on J_k is of the form $\alpha_k r^2 + \beta_k$, yielding

$$s(\underline{x}) \le \sum_{k=1}^{n} \alpha_k ||\underline{x} - v_k||^2 + \sum_{k=1}^{n} \beta_k + p(\underline{x}),$$

for $\underline{x} \in I$. Since, for most RBFs, the polynomial p has degree at most two, the upper bound is a bivariate quadratic function, obtained by adding the coefficients of the upper bounds for each individual summand. Moreover, the maximum value $U(I)$ of this upper bound on the interval I is easily computed. Due to lack of space we just mention that for most RBFs the coefficients α_k and β_k are determined in a rather straightforward way by solving a simple optimization problem, once for each type of RBF. A quadratic lower bound for the RBF-interpolant s on I is determined similarly. In view of the special shape of the quadratic upper and lower bounds this approach is called the *bounding paraboloid strategy (BPARAB)*

Computing $\Box\nabla s(I)$. To find optimal ranges $\Box s_x(I)$ and $\Box s_y(I)$ for the components of the gradient of the RBF-interpolant (1), first note that s_x is given by

$$s_x(\underline{x}) = \sum_{k=1}^{n} w_k \frac{\varphi'(||\underline{x} - v_k||)}{||\underline{x} - v_k||}(x - v_{kx}) + p_x(\underline{x}), \quad (2)$$

where $\underline{x} = (x, y)$ and $v_k = (v_{kx}, v_{ky})$. Applying the same approximation strategy as before we find quadratic lower bounds on the one-dimensional interval J_k for each of the *univariate factors* $w_k\,\varphi'(r)/r$, leading to a bivariate *cubic* lower bound $L_k(\underline{x})$ on the box I for the k-th summand in (2) of the form

$$L_k(\underline{x}) = a_k\,x\,(x^2 + y^2) + Q_k(\underline{x}),$$

where a_k is a real constant, and $Q_k(\underline{x})$ is a quadratic polynomial. A cubic upper bound $U_k(\underline{x})$ of this form is found similarly. A straightforward derivation yields the minimal value of $\sum_k L_k(\underline{x}) + p_x(\underline{x})$ and the maximal value of $\sum_k R_k(\underline{x}) + p_x(\underline{x})$ on I, and, hence, a good interval $\Box s_x(I)$. A good interval $\Box s_y(I)$ is computed similarly.

As we will show in Section 4, this strategy improves the performance of the certified meshing algorithm ApproximateCurve considerably for various RBFs.

Bounding plane strategy for the cubic RBF. The cubic RBF, given by $\varphi(r) = r^3$, corresponds to the triharmonic spline in 3D, which is used widely in reconstruction of geometric surfaces from scattered point samples. Therefore, for this case we tried to design an even better strategy based on special properties, like convexity, of the RBF. More precisely, using well-chosen *linear* upper and lower bounds, we were able to improve the running time even further in some cases. For experiments with this *bounding plane strategy (BP)*, corroborating this improvement, we refer to Section 4.

4 Experimental results

We present some 2D experiments with algorithm ApproximateCurve, implementing range functions based on IA, AA, BPARAB and, for the cubic RBF, the BP-strategy. We extract the zero sets of various RBF-interpolants, and compare the number of leaves (NOL) of the subdivision tree and the CPU-time (CPU). The RBF-interpolants are constructed using uniform sample interpolation points extracted from several well-known functions, cf. [6], over a bounded domain.

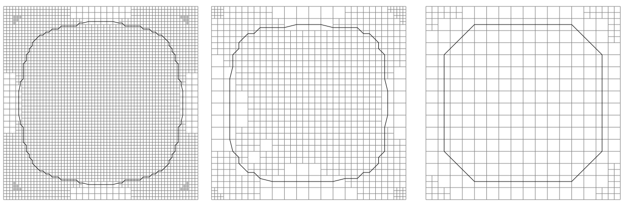


Figure 2: Isocurve extraction for a cubic RBF-interpolant (100 centers) of the function $xy(x-1)(y-1)-0.02$, sampled uniformly on the square $[0,1]\times[0,1]$ using strategies: (i) AA , (ii) BP and (iii) BPARAB.

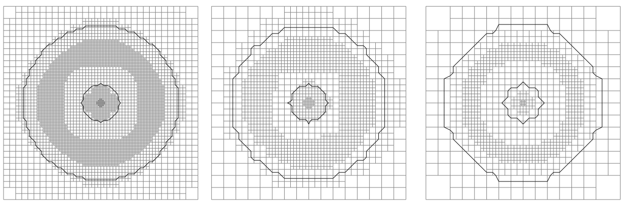


Figure 3: Isocurve extraction for a cubic RBF-interpolant (100 centers) of the function $(x^2+y^2)(1-\sqrt{x^2+y^2}) - 0.04$, sampled uniformly on the square $[-1.2,1.2]\times[-1.2,1.2]$ using strategies: (i) AA , (ii) BP and (iii) BPARAB.

We used the `Boost` library [1] for IA, and the library [2] for AA. All experiments have been performed on a 3GHz Intel Pentium 4 machine under Linux with 1 GB RAM using the `g++` compiler, version 3.3.5.

Experiments with Cubic RBF. Our first sequence of experiments has been performed using cubic-based interpolants. In other words, we used the RBF given by $\varphi(r) = r^3$. Tables 1–3 presents the measured performance for different optimization strategies. Figure 2–4 contain the corresponding isocurves, together with the boxes corresponding to the leaf-nodes of our subdivision tree.

Note that Table 1 shows that straigthforward use of IA does not lead to convergence (in reasonable time), except in trivial cases. Therefore, we discard IA from our remaining experiments.

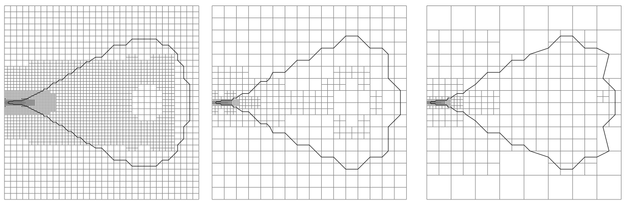


Figure 4: Isocurve extraction for a cubic based RBF-interpolant (100 centers) of the function $4y^2 - (x+1)^3(1-x)$, sampled uniformly on the square $[-1.1,1.1]\times[-1.1,1.1]$ using strategies: (i) AA , (ii) BP and (iii) BPARAB.

Experiments with Multiquadric RBF. Next, we show some more experimental results using the multiquadric RBF given by $\varphi(r) = \sqrt{1+r^2}$. Table 4 compares the performance of the AA and BPARAB strategies for this case. Figures 5 and 6 contain the corresponding isocurves.

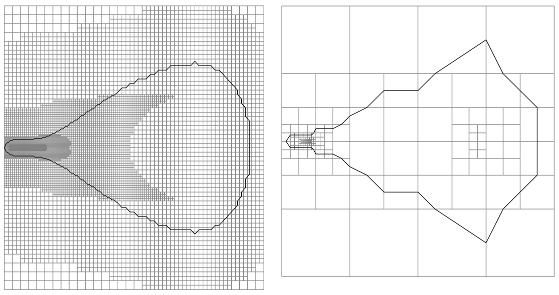


Figure 5: Isocurve extraction for a multiquadric RBF-interpolant (49 centers) of the function $4y^2 - (x+1)^3(1-x)$, sampled uniformly on the square $[-1.1,1.1]\times[-1.1,1.1]$ using strategies: (i) AA and (ii) BPARAB.

Conclusion. Our experiments show that IA has unacceptable performance, that AA converges in most experiments with the cubic RBF but fails for the multiquadric-based interpolants, that BPARAB is a general and fast method, and that the BP-strategy for cubic RBFs does not perform better than BPARAB.

References

[1] `Boost` interval arithmetic library. `www.boost.org`.

	IA		AA		BP		BPARA	
NOC	NOL	CPU	NOL	CPU	NOL	CPU	NOL	CPU
25	138616	6.54s	1264	1.6s	280	0.22s	154	0.15s
49	484660	39.6s	2140	5.3s	325	0.49s	274	0.49s
100	1726852	4m13s	3856	25.8s	898	2.19s	304	1.19s
225	6757600	36m47s	5848	1m56s	1156	8.74s	769	6.30s
400	-	-	12880	9m11s	2008	19.1s	1081	16.5s
625	-	-	16408	27m59s	3676	56.6s	1120	27.6s
900	-	-	17980	629s	4237	1m46s	1636	49.5s
1156	-	-	19084	98m55s	4435	2m39s	2032	1m11s

Table 1: Space and time complexity corresponding to Figure 2.

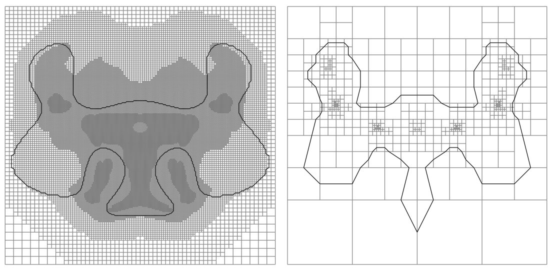


Figure 6: Isocurve extraction for a multiquadric RBF-interpolant (25 centers) of the function $(y-x^2+1)^4+(x^2+y^2)^4-1=0$, sampled uniformly on the square $[-1.2,1.2]\times[-1.4,1.0]$ using (i) AA and (ii) BPARAB.

	AA		BP		BPARAB	
NOC	NOL	CPU	NOL	CPU	NOL	CPU
25	934	1.29s	181	0.152s	142	0.152s
49	1810	5.96s	376	0.66s	337	0.63s
100	3592	25.9s	784	2.77s	589	2.27s
225	7072	2m39s	1483	11.7s	1183	10.1s
400	11956	2m24s	2350	32.3s	1492	23.2s
625	18766	34m	3691	1m15s	2104	51.2s
900	25252	88m	5122	2m30s	3136	1m45s
1156	27910	154m	5668	3m25s	3895	2m39s

Table 3: Space and time complexity corresponding to Figure 4.

	AA		BP		BPARAB	
NOC	NOL	CPU	NOL	CPU	NOL	CPU
25	2908	4.07s	640	0.51s	448	0.388s
49	6820	16.5s	1237	1.51s	580	1.10s
100	9052	55.7s	1741	4.10s	1156	3.70s
225	18808	5m36s	3580	19.8s	1528	10.5s
400	24988	17m16s	4492	41.8s	1972	29.0s
625	31492	46m57s	5338	1m9s	3652	1m10s
900	34888	108m	6565	2m6s	4120	1m46s
1156	37288	198m	7663	3m53s	4540	2m40s

Table 2: Space and time complexity corresponding to Figure 3.

	AA		BPARA	
NOC	NOL	CPU	NOL	CPU
25	1216	1.48s	52	0.068s
49	7726	20.8s	154	0.30s
100	115312	13m36s	274	1.12s
225	—	—	289	2.71s
400	—	—	520	8.66s
625	—	—	598	15.8s
900	—	—	610	22.7s
1156	—	—	874	41.6s

Table 4: Complexity of Multiquadric-based meshing corresponding to Figure 5.

[2] `C++` affine arithmetic library. `savannah.nongnu.org/projects/libaffa`.

[3] J.-D. Boissonnat, D. Cohen-Steiner, and G. Vegter. Isotopic implicit surface meshing. *Discrete and Computational Geometry*, 39:138–157, 2008.

[4] L. H. de Figueiredo and J. Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms.*, 00:1–13., 2003.

[5] A. Iske. Scattered data modelling using radial basis functions. In A. Iske, E. Quak, and M. S. Floater, editors, *Tutorials on Multiresolution in Geometric Modelling*, Mathematics and Visualization, pages 287–315. Springer-Verlag, Heidelberg, 2002.

[6] H. Lopes, J. Oliveria, and L. Figueiredo. Robust adaptive polygonal approximation of implicit curves. *Computers and Graphics*, 2002.

[7] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics (Proceedings SIGGRAPH 1987)*, 21(Annual Conference Series):163–169, 1987.

[8] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang. Comparison of interval methods for plotting algebraic curves. *Comput. Aided Geom. Des.*, 19(7):553–587, 2002.

[9] S. Plantinga and G. Vegter. Isotopic meshing of implicit surfaces. *The Visual Computer*, 23:45–58., 2007.

[10] B. Stander and J. Hart. Guaranteeing the topology of an implicit surface polygonizer for interactive modeling. In *Proceedings SIGGRAPH*, pages 279–286, 1997.

Low-resolution Surface Mapping: a Topological and Geometrical Approach

Jean-Marie Favreau[*], Vincent Barra[†]

Abstract

In this paper we tackle the problem of mapping a surface acquired from a real object onto a piece of the plane, taking into account the topological and geometrical properties of the surface, as well as the specificity of low-resolution acquisitions. We first describe the general background of surfaces unfolding and cutting. We then introduce the cutting process used to manage the topological constraint of the one-to-one mapping, with some speedup improvements. Specific geometrical constraints linked to the low resolution context are detailed, and some experimental results on real and synthetic data are finally presented.

1 Introduction

There are several ways to produce 3D acquisitions from a real object. They can mainly be separated into volumic acquisitions, *e.g.* in medical imaging, and point clouds generated from 3D scanners or cameras. Both of the data structures are usually turned into surfaces described by triangles, using isosurface reconstruction methods that ensure a topologically correct reconstructed surface [8], or a sign distance [3]. They can also be fitted with analytical surfaces such as RBF [1]. Due to the precision and resolution, the resulting virtual surface may not capture the full topological and/or geometrical details of the original surface, and are prone to artifacts that misrepresent the data. These errors are common for specific geometric configurations, producing non-existing junctions or handles, disjunctions or hole fillings.

To solve these topological artifacts, volumic approaches can be proposed, such as the use of skeletons [11]. This method thus can only manipulate volumic data, and cannot be influenced by geometrical properties like local curvature. On the other hand, surfacic approaches [10] remove small tunnels, but do not manage geometrical properties of surfaces, like geodesic distances or local curvature. In this paper, the specificity of these surfaces is managed during the manipulation steps, taking into account both topological and geometrical properties of the surfaces.

[*]Université Blaise Pascal / LIMOS UMR 6158 Jean-Marie.Favreau@isima.fr

[†]Université Blaise Pascal / LIMOS UMR 6158 Vincent.Barra@isima.fr

2 Background

2.1 Topological definitions

Let X and Y be two topological spaces. X and Y are said to be *homeomorphic* if there is an homeomorphism between these spaces, *i.e.* if there is a bijective continuous function $f : X \to Y$, with a continuous inverse function f^{-1}. A 2-*manifold* or surface $\mathcal{M}$ without boundary is a topological space locally homeomorphic to a disc. A surface with boundary can also be locally homeomorphic to an half disc. A *non-separating curve* in $\mathcal{M}$ is a curve that can be drawn on the surface without separating it. A *simple closed curve* in $\mathcal{M}$ is a curve homeomorphic to a circle.

The *genus* g of a surface $\mathcal{M}$ is defined as the largest number of non-intersecting non-separating simple closed curves.

2.2 2-mesh definition

In the following, we assume that the first process of surface reconstruction has been applied, producing a 2-*mesh* $\mathcal{M}$, defined as a *2-manifold simplicial 2-complex* (with boundaries or not), *i.e.* $\mathcal{M}$ is an union of 0-simplices called vertices, 1-simplices called edges and 2-simplices called triangles, and locally homeomorphic to either a disc or an half disc. We call *a boundary* each connected component of the simplicial 1-complex defined by the boundary of $\mathcal{M}$. Surfaces are usually classified by their genus and the number of their boundaries.

In the following, we assume that the triangulated surfaces are 2-meshes, *i.e.* without local artifacts.

2.3 Mapping a surface

An usual approach to visualize the whole surface of a 2-mesh $\mathcal{M}$ is to map it to a piece of the plane. It is also a natural approach of UV mapping for texture drawing on complex surfaces. We define a *mapping* between a 2-mesh $\mathcal{M}$ and a piece of the plane as an one-to-one continous function $f : \mathcal{M} \to \mathbb{R}^2$. This definition does not allow foldings, and mappings thus only exists for surfaces homeomorphic to a piece of the plane.

Mapping algorithms can be subdivided into fixed-border and free-border methods [7]. Fixed-border methods allow the boundary to be constrained. The resulting map does not take into account the shape of the boundary, and does not preserve angles or distances. Free-border methods produce only locally bijective mappings: the boundary of the unfolded surface may contain self-crossings on the plane. However, these self-crossings do not forbid a local use of the mapping and if neccesary, local arrangements of the mapping may be applied [9].

For practical purposes, the majority of the methods imposes that $\mathcal{M}$ is homeomorphic to a disc. In section 3, this topological constraint has been relaxed using a cutting step.

2.4 Cuttings using paths and loops

Most of the manipulated surfaces are not homeomorphic to a piece of the plane. Unfolding the surface thus make some preprocessings mandatory, reducing the genus and introducing boundaries. These transformations have to be as unnoticeable as possible. The most intuitive method to produce such a transformation is the use of cuttings along paths. A 1-*path* of a 2-mesh $\mathcal{M}$ is defined as a list of n oriented edges $s_0, ..., s_{n-1}$ where for each $i = \{0, ..., n-2\}$, the terminal vertex of s_i is the initial vertex of s_{i+1}. A 1-*cycle* of a 2-mesh $\mathcal{M}$ is defined as a 1-path $s_0, ..., s_{n-1}$ where the initial vertex of s_0 is the terminal vertex of s_{n-1}. A cutting along a 1-path p is naturally performed by duplicating vertices and edges of p, thus removing junctions between triangles adjacent to p.

3 Topological constraints

Generally speaking, the mapping methods compute a transformation from a surface of genus 0 with at least one boundary to a piece of the plane. In practice, methods described in section 2.3 unfold surfaces homeomorphic to a disc, *i.e.* with exactly one boundary. A preliminary topological transformation step is thus needed to be able to apply those unfolding methods.

3.1 Cutting the surface

Let $\mathcal{M}$ be a 2-mesh of genus g with b boundaries. As mentioned in section 2.4, the topological transformations have to be as unnoticeable as possible, and cutting is a solution that reduces the genus and the number of boundaries with only small modifications. In this context, we call *cutting* C of a 2-mesh $\mathcal{M}$ a set of edges whose complement in $\mathcal{M}$ is homeomorphic to a disc.

Some algorithms have already been proposed to produce such a cutting. Colin de Verdière *et al.* described a topological algorithm [2] producing a minimal cutting in terms of number of 1-paths. However, this cutting only focuses on topological properties of the 2-mesh, and does not handle the geometry. Erickson and Har-Peled [5] computed an almost minimal cutting in terms of cutting's length, using a distance d on edges. In our application, this method seems to be the most interesting approach, since it manages topology without ignoring the geometry of the 2-mesh. In [5], authors defined a $o(n \log(n))$ algorithm to compute the shortest non-separating 1-loop (1-nsloop) starting from a given point b, called basepoint. The shortest 1-path and 1-loop are computed using the Dijkstra algorithm [4]. Algorithm 1 describes the global cutting algorithm:

```
**Data**: 2-mesh $\mathcal{M}$ non homeomorphic to a sphere
**while** *genus of* $\mathcal{M}$ $!= 0$ **do**
    $c$ = shortest 1-nsloop;
    Cut $\mathcal{M}$ according to $c$;
**while** *number of boundaries of* $\mathcal{M}! = 1$ **do**
    $c$ = shortest 1-path between 2 boundaries;
    Cut $\mathcal{M}$ according to $c$;
```

Algorithm 1: Cutting a surface into a disc

We used an alternative method to build a better approximation of the optimal cutting, using Algorithm 2 to compute the shortest 1-nsloop:

```
**Data**: 2-mesh $\mathcal{M}$ non homeomorphic to a sphere
**Result**: The shortest 1-nsloop $l$
$B$ = a complete set of basepoints;
**foreach** $p \in B$ **do**
    $l_p$ = shortest 1-nsloop containing $p$;
    **if** *first step* **or** *length*$(l_p)$ < *length*$(l)$ **then**
        $l \leftarrow l_p$
```

Algorithm 2: Finding the shortest 1-nsloop

The main difference between the original method and our approach is the set of basepoints B of $\mathcal{M}$. In [5], this set was reduced to optimize the complexity, but with a loss of precision. In our approach, B is a set of points such as if a 1-nsloop exists on $\mathcal{M}$ it should have a point contained in B. Basepoints can be computed using a growing surface process: a point p is first selected on the interior of $\mathcal{M}$. Triangles of the 2-mesh are then added as a growing surface, preserving the original connectivity and assuming that this wavefront is continually homeomorphic to a disc.When the wavefront stops, some boundaries of the resulting surface may not be boundaries of $\mathcal{M}$, and the set of their points is equal to B.

Our approach has a complexity $O(gnm \log n)$, where m is the number of basepoints and n is the number of points of $\mathcal{M}$. Some enhancements of this algorithm can then be considered to improve the practical running time, more particularly focusing on the computation of the shortest 1-nsloop.

3.2 Using a bounded Dijkstra algorithm

Algorithm 2 computes for each basepoint p the corresponding shortest 1-loop, and compares its length to the length of the shortest already computed 1-loop l. Paths longer than l are ignored. Moreover, the shortest 1-loop computation from p is mainly based on a Dijkstra algorithm: points

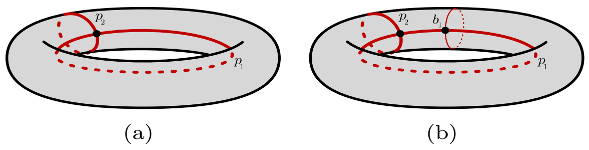


Figure 1: The potential good basepoint selection using shortest paths. Fig. (a): The selected basepoints, structured into two paths. Fig. (b): The potential good basepoint b_1 selected using the shortest path between the two sides of p_1.

are ordered according to the result of this algorithm to grow the wavefront.

Using l as an additional piece of information from a given basepoint, the computation may then be rewritten as follows: *Compute the shortest* 1-*nsloop* l_p *containing* p *and shorter than length*(l). This truncates the Dijkstra algorithm and the wavefront growing stage.

The complexity of this modified 1-loop computation is less than $o(m(\log(n) + l_l \log(l)))$ where l_l is the higher length of the shortest 1-loop starting from a basepoint of B. For practical purposes, this maximum bound is rather small and this improvement clearly speeds up the 1-loop computation, as illustrated in Table 1.

If this improvement dramatically increases the shortest 1-nsloop computation, it often depends on the order of the basepoints: the sooner the shortest 1-loops are founded, the sooner next searches are truncated, increasing their computation time. We select the first basepoint from each path p of B using the shortest path from one side of p on the complementary of B to the other (Figure 1).

3.3 Using previous cuttings

Algorithm 1 uses successively several calls of Algorithm 2. After each computation of the shortest 1-loop, the 2-mesh is cut according to this result. The length of the ist 1-loop, $i > 1$ will thus be greater or equal than the length of the $(i-1)$st 1-loop. During the construction of the ist 1-loop, and when a junction of the growing surface occurs, the length of the 1-loop corresponding to this junction may thus be computed before computing the number of connected components of the complement of the growing surface. If this length is greater than the length of the $(i-1)$st 1-loop, there is no need to compute the number of connected components since this new 1-loop is not a non separating 1-loop.

4 Geometrical constraints

The cutting method (section 3.1) considers only a part of the geometry by the shortest geodesic distances. Much more geometrical properties of the surface can be handled by enhancing this original method.

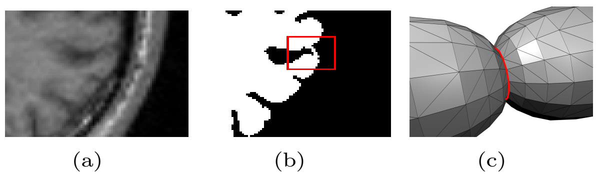


Figure 2: Fig. (a), (b): A topological error (junction between two parts of the surface) after segmentation step. Fig. (c): A cutting path (drawn in red) on the non-zero local curvature area.

4.1 Patching the surface

Surfaces mainly originate from low resolution acquisitions. In this context, some topological errors are common for specific geometric configurations. If partial volume effects, undersampling or interpolation artifacts occurs in the original data, some junctions between parts of the surface may be observed (Figure 2 (a), (b)).

To remove these errors, we used a variation of the topological and geometrical algorithm previously described. Given a threshold length m, this algorithm cuts all the junctions with perimeter smaller than m and patches the corresponding boundaries by adding small discs to close it. The original boundaries of $\mathcal{M}$ with perimeter smaller than m are also patched.

When used before Algorithm 1, this method manages the specificity of the low resolution surfaces and creates some allowable unfolding, with few overlaps (Figure 3(c)).

4.2 Using non-Euclidean distances

The cutting and patching algorithms both use Dijkstra algorithm to select the 1-loops. Thus the only requirement is a non-negative cost function on the edges, seen as a pseudo-distance d (section 3.1). The obvious cost function is the Euclidean distance d_E between two vertices in $\mathbb{R}^3$, but other choices, guided by application, are also possible.

During the patching process, the algorithm looks for surface junctions. In many applications like medical imaging, junctions are cut along a valley (Figure 2(c)). Each edge e of the cutting path takes part in two triangles, and the local curvature can be described by the angle $\alpha(e) \in [-\pi, \pi]$ between their normals. The Euclidean distance may then easily be modified to address this local curvature. Let $c \in [0,1]$, we define $d_c(e) = d_E(e)(1 - c\frac{|\alpha(e)|}{\pi})$, for each edge e of $\mathcal{M}$. The multiplicative function $m : e \mapsto 1 - c\frac{|\alpha(e)|}{\pi}$ has the good following property: if $\alpha(e) = 0$, e takes part in a flat area, and $m(e)$ should reach

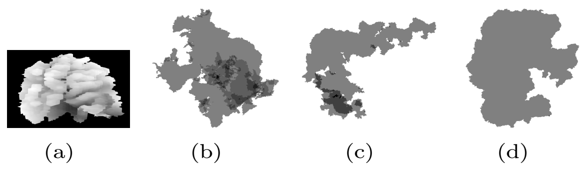


Figure 3: Fig. (a): A neighbourhood of a central sulcus extracted from the original volumic data [6]. Fig. (b), (c), (d): Cortical maps unfolded by Circle Packing after cutting steps (naïve, topological, and patching + topological). Dark grey shows regions of overlappings.

Enhancements	Brain (a)	Synthetic (b)
none	$>$ 5 h 7 mn	7.68 s
sect. 3.2	720 s	0.84 s
all (section 3)	548 s	0.81 s

Table 1: Computation time (on a 2.6 Ghz PC) of the first step of Algorithm 1 according to the improvements of section 3, (a) on a cortical surface [16,057 points, 31,705 triangles], a genus of 56 and 7 boundaries; (b) on a synthetic non-boundary surface of genus 4 [2,202 points, 4,416 triangles].

the maximum. Otherwise, the highest is $|\alpha(e)|$, the smallest d_c. d_c thus grants the paths along paths with high local curvature, in particular in the junctions.

5 Results

A `C++` implementation of these modified algorithms has been performed to illustrate their performing on low resoluted surfaces. The method has also been applied on classical surfaces, notably with high genus and structures similar to the junctions of the low resolution acquisitions.

In medical imaging, an unfolding approach has been described in a previous work [6], to produce a flattened map of the region of interest on the cortical surface, using T1-weigthed Magnetic Resonance Imaging scans. After some segmentation steps and a surface reconstruction, a 2-mesh is optained, modeling the region of interest on the cortical surface. The cutting method used in [6] did not take into account geometry, producing maps with overlaps for high genus original surfaces (Figure 3(b)). The approach described in this paper was applied to the 2-meshes stemming from the original data.

The resulting surface was then unfolded using classical unfolding tools. Figure 3(c) and 3(d) illustrate the quality of the generated maps. The cortical map generated using a patching step before the cutting step is clearly more satisfactory.

We applied the cutting method on the cortical mesh using the various improvements of the cutting algorithm (section 3). Table 1 compiles the computation time of the first part of Algorithm 1 using these improvements. If the evaluation of the theoretical computation time of our improvements is hard, Table 1 nevertheless illustrates the significant time-saving observed with our implementation.

6 Conclusion

We presented here a topological and geometrical approach to cut the surface before mapping it onto the plane. By improving the complexity of the cutting algorithm, we optained a useful tool that produces maps without coverings, allowing the user to navigate using the 2D map on the 3D surface thanks to the bijective mapping.

References

[1] J. C. Carr, R. K. Beatson, et al. Reconstruction and representation of 3d objects with radial basis functions. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 67–76. ACM Press / ACM SIGGRAPH, 2001.

[2] É. Colin de Verdière and F. Lazarus. Optimal system of loops on an orientable surface. *Discrete & Computational Geometry*, 33:507–534, 2005.

[3] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, 1996.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[5] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 244–253, New York, NY, USA, 2002. ACM.

[6] J.-M. Favreau, S. Hemm, C. Nuti, et al. A tool for topographic analysis of electrode contacts in human cortical stimulation. In *MMBIA'07*, Rio de Janeiro, Brazil, October 2007. ICCV 2007.

[7] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[8] J.-O. Lachaud. Topologically defined isosurfaces. In *DCGA '96: Proceedings of the 6th International Workshop on Discrete Geometry for Computer Imagery*, pages 245–256, London, UK, 1996. Springer-Verlag.

[9] A. Sheffer and E. de Sturler. Parameterization of Faceted Surfaces for Meshing using Angle-Based Flattening. *Engineering with Computers*, 17(3):326–337, 2001.

[10] Z. Wood and I. Guskov. Topological noise removal. *Proceedings of Graphics Interface*, pages 19–26, 2001.

[11] Q.-Y. Zhou, T. Ju, and S.-M. Hu. Topology repair of solid models using skeletons. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):675–685, 2007.

Matching Terrains under a Linear Transformation[*]

Pankaj K. Agarwal[†] Boris Aronov[‡] Marc van Kreveld[§] Maarten Löffler[§] Rodrigo I. Silveira[§]

Abstract

We study the problem of matching two polyhedral terrains, where one can be changed vertically by a linear transformation of the third coordinate (scaling and translation). We give an algorithm that minimizes the maximum distance over all linear transformations in $O(n^{4/3}\text{polylog } n)$ expected time. We also study matching two 1-dimensional terrains, and give a $(1+\varepsilon)$-approximation algorithm for minimizing the area in between that runs in $O(n/\sqrt{\varepsilon})$ time, for any fixed $\varepsilon > 0$.

1 Introduction

In GIS, many data sets are bivariate scalar functions, for example annual precipitation, nitrate concentration, depth to groundwater, and elevation above sea level. Such functions are either stored by a regular square grid of values, or by a triangulation over some domain. In the case of a triangulation, one usually assumes linear interpolation over the values stored at the vertices. This gives a polyhedral terrain, that is, an xy-monotone polyhedral surface in $\mathbb{R}^3$.

Often, data sets for the same region but with a different theme are not independent: if there is more precipitation, the soil humidity will be higher, and if hill slopes are steeper, there will be more soil loss per year. Such relations can be analyzed by matching the representations of two data sets.

In this paper we assume that two bivariate scalar functions over the same domain are given as (polyhedral) terrains. We want to determine if there is a linear transformation applied to one to make it match well with the other. The linear transformation applies only to the scalar values, not to the world coordinates. Hence, a linear transformation has one scaling component s and one translation component t.

Given two terrains $f, g : D \to \mathbb{R}$, where D is a bounded, triangulated subset of $\mathbb{R}^2$, our goal is to find a linear relation between the two terrains. That is, we want to find values $s, t \in \mathbb{R}$ such that $sf + t$ matches g as well as possible. To compare the two terrains, we use two different measures:

$$\mu(s,t) = \max_{x,y} |sf(x,y) + t - g(x,y)|, \text{ and} \quad (1)$$

$$\mu(s,t) = \int_{x,y} |sf(x,y) + t - g(x,y)| dxdy. \quad (2)$$

With each measure, we want to minimize its value over s and t, namely

$$\min_{s,t} \mu(s,t).$$

In what follows we denote by n the total number of vertices in f and g, and sometimes use the O^* notation, which ignores polylogarithmic factors.

We say that f and g are *aligned* if their projections on the (x,y)-plane are the same, that is, they have exactly the same triangulation.

2 Minimizing the max distance of two 2-d terrains

In this section the goal is to find s, t to minimize $\mu(s,t)$ according to Eq. (1), namely, to minimize the maximum vertical distance between g and $sf + t$.

We begin by observing that, when the terrains are aligned, the maximum of μ is always obtained at a vertex of the terrain. When the terrains are not aligned, the maximum can occur either at a vertex of one of the terrains or at the intersection of two edges in the xy-projection. This implies that in both cases the problem is inherently discrete.

Let v be a vertex of one of the terrains or the intersection point of two terrain edges (always in the xy-projection). Let $d_v(s,t) = |sf(v) + t - g(v)|$, where $f(v)$ (resp. $g(v)$) gives the height of v in f (resp. g). Thus $d_v(s,t)$ defines a surface in the (s,t,μ)-space (here μ stands for the possible values for the objective function). This surface consists of two halfplanes with a V-shape (due to the absolute value). The collection of the halfplanes from all vertices gives an arrangement of planes where $\mu(s,t)$ can be viewed as the upper envelope of the planes. Since we are interested in the upper envelope only, we can consider the halfplanes to be full planes.

[*]This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503, and also through the project GOGO. Work by B.A. has been partially supported by a grant from the U.S.-Israel Binational Science Foundation, by NSA MSP Grant H98230-06-1-0016, and NSF Grant CCF-08-30691.

[†]Department of Computer Science, Duke University, `pankaj@cs.duke.edu`

[‡]Department of Computer Science and Engineering, Polytechnic Institute of NYU, `http://cis.poly.edu/~aronov`

[§]Department of Information and Computing Sciences, Utrecht University, {`marc,loffler,rodrigo`}`@cs.uu.nl`

It follows that the lowest value of μ can be computed using linear programming (LP) in three dimensions. For the case of aligned terrains, each terrain vertex generates two planes, resulting in a system with $O(n)$ inequalities. Standard methods for linear programming in fixed dimension allow to solve the problem in $O(n)$ time [4].

When the terrains are not aligned, the same approach can be used if all the intersection points between terrain edges are considered vertices. However, their number may become $\Omega(n^2)$, hence solving the linear program would take quadratic time. Note, however, that we have $O(n)$ running time for non-aligned *fat* terrains, since the number of intersections is linear for two such terrains [6]. In what follows we show how to reduce the potentially quadratic running time for arbitrary terrains by considering only a linear number of intersection points. From now on we will refer to both the terrain vertices and the intersection points in the xy-projection as *vertices*.

We will avoid computing the quadratic set of constraints explicitly by applying a random sampling approach. Our method is similar to the LP algorithm in [5], and is sketched in Algorithm 1. Recall that each vertex defines two constraints.

Algorithm 1 MinMaxDistance

```
 1: $R \leftarrow$ constraints induced by random subset of $3n$
    vertices of the projection
 2: repeat
 3:   Solve LP problem constrained to $R$
 4:   $V \leftarrow$ set of violated constraints
 5:   if $|V| > 2n$ then
 6:     $R \leftarrow$ constraints induced by a new random
        subset of $3n$ vertices of the projection
 7:   else
 8:     $R \leftarrow R \cup V$
 9:   end if
10: until $V = \emptyset$
11: return Solution from step 3
```

Several parts of the algorithm are explained in more detail below.

First a random subset of the vertices must be chosen. This is done in any way that guarantees that all subsets with r elements have the same probability of being chosen. This can be done using the two-level hereditary segment tree mentioned below.

Then the linear program that has the constraints induced by the vertices in R is solved. This is a 3-dimensional LP with $O(n)$ constraints that can be solved in $O(n)$ time using standard algorithms [4]. The solution gives us concrete values of s_0, t_0 and μ_0. That is, after that we have two concrete terrains, g and $f_0 = s_0 f + t_0$.

Computing the violated constraints. Since the solution (s_0, t_0) from line 3 only considers the $O(n)$ constraints in R, some of the $O(n^2)$ original constraints may be violated by this solution. A violated constraint implies that there is some point (original vertex or intersection point) in g that is at a vertical distance larger than μ_0 from f_0. To find the violated constraints we need to find the vertices at distance larger than μ_0 from the other terrain. The distances between vertices of f_0 and triangles in g, or the other way around, can easily be computed in $O(n \log n)$ time. More challenging is finding the pairs of edges with vertical distance larger than μ_0, since we want to report these constraints only if there are at most $2n$ of them, without spending more time otherwise.

To report this type of violated constraint we proceed in two steps. First we use a data structure based on hereditary segment trees to reduce the vertical distance problem between line segments to a problem of reporting lines in 3D that are at least μ_0 apart. Then we solve that problem by mapping the lines in space to Plücker points and Plücker hyperplanes, and solving a halfspace range reporting problem in 5D projective space. By using the trade-off techniques for halfspace range reporting we find all the violated constraints in $O^*(n^{4/3} + |V|)$ time, with the option to stop reporting if $|V| > 2n$. A more detailed description follows.

For the first part, we use two-level hereditary segment trees [3]. We consider the set of segments of the projection of the two terrains. They can be seen as a set of red and a set of blue segments (one set from each terrain; segments from the same terrain cannot intersect). As in [3], the segment tree can be augmented to produce a bipartite clique decomposition of the intersecting pairs of edges. Within each clique of segments, we extend the line segments to lines, without adding or missing any intersection/distance. The complete data structure can be built in $O(n \log^2 n)$ time.

Next we must find the pairs of red-blue lines that are more than μ_0 vertically apart. This second problem is transformed into a halfspace range reporting problem as follows. Testing whether two lines ℓ_1, ℓ_2 in 3D lie at a vertical distance of more than μ_0 from each other can be formulated (after translating one of the lines by μ_0) as testing whether the Plücker point of ℓ_1 lies in one of the halfspaces bounded by the Plücker halfplane of ℓ_2. We map the lines from, say, g, to Plücker points and perform a halfspace reporting query with each of the halfspaces bounded by the Plücker hyperplanes of the lines induced by edges of f_0. To solve the halfspace reporting problem we apply standard trade-off techniques for geometric range searching (see for example [1]). We build a data structure of size $O^*(n^{4/3})$ that allows to answer halfspace reporting queries in $O(n^{1/3} + k)$ time, where k is the output size. In our context, k is the number of pairs of segments with distance more than μ_0. Note that in line 4 of the algorithm, we do not need to report all violated constraints, but we can stop when their number exceeds $2n$. Thus line 4 takes $O(n^{4/3})$ time.

Choosing a new random sample. The standard random-sampling argument in the Clarkson-Shor framework (see [5]) states that if we choose a random sample of size r among m constraints, and solve the LP with only those constraints, the expected number of violated constraints will be roughly dm/r, where d is the dimension of the LP problem. In our case we have $d = 3$, $r = 3n$ and $m \approx n^2$. Thus the expected number of violated constraints is approximately n, and on average the random subset of constraints will have to be generated twice.

Number of iterations. It can be shown that once $|V|$ has fewer than $2n$ constraints, the loop of Algorithm 1 will be executed at most four times. This is because every time the LP problem is solved, one of the three constraints that define the optimal solution is added to the set of violated constraints (see [5] for a proof), and from then on, it will be part of R. Therefore after executing line 3 at most three times, the set of constraints R will contain the three vertices that define the optimum, hence in the next (fourth) iteration the optimum will be found.

Theorem 1 *Given two terrains with n vertices, a linear transformation minimizing the maximum distance between the terrains can be found in $O^*(n^{4/3})$ expected time.*

3 Minimizing the area between two 1-d terrains

Let f, g be two 1-dimensional terrains, or, the graphs of two piecewise linear functions. This section discusses the problem of aligning f and g in order to minimize the area in between. For any value of s and t, let $\mu(s,t)$ denote the area between $sf + t$ and g over their domain. We will study the problem in the solution space, the (s,t)-plane.

Analytic form of μ. The bivariate function $\mu(s,t)$ has an analytic form that depends on which edges of $sf + t$ and g intersect. If $x_1, \ldots, x_n$ is the increasing order of the x-coordinates of all vertices of f and g, then we may as well assume that both f and g have vertices at these x-coordinates. At worst, this doubles the number of vertices of the input. Let $(x_1,a_1), \ldots, (x_n,a_n)$ be the vertices of f and let $(x_1,b_1), \ldots, (x_n,b_n)$ be the vertices of g.

Consider any vertical slab defined by two consecutive x-coordinates of vertices x_i and x_{i+1}, see Figure 1. Let f_i, g_i, and μ_i be the functions f, g, and μ restricted to the domain $[x_i, x_{i+1}]$. Unless f_i is constant,

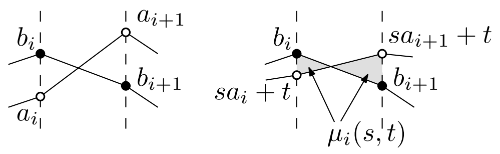


Figure 1: An edge of f and an edge of g, and the function $\mu_i(s,t)$ illustrated.

there is exactly one pair s,t that makes $sf_i + t \equiv g_i$, and $\mu_i(s,t) = 0$. This happens when both $sa_i + t = b_i$ and $sa_{i+1} + t = b_{i+1}$.

Lemma 2 *The region in the (s,t)-plane where $sf_i + t$ and g_i intersect is a double wedge whose bounding lines are $\ell_i \,:\, t = -a_i s + b_i$ and $\ell_{i+1} \,:\, t = -a_{i+1}s + b_{i+1}$. These lines intersect in the apex $(\bar{s}_i, \bar{t}_i)$, where $\bar{s}_i f_i + \bar{t}_i \equiv g_i$ and $\mu_i(\bar{s}_i, \bar{t}_i) = 0$.*

Above ℓ_i and ℓ_{i+1} (in the direction of t), μ_i is linear in s and t, and the same is true below ℓ_i and ℓ_{i+1}. Below ℓ_i and above ℓ_{i+1} in the (s,t)-plane, we have

$$\mu_i(s,t) = (x_{i+1} - x_i) \times \frac{(b_i - sa_i - t)^2 + (sa_{i+1} - b_{i+1} + t)^2}{2(sa_{i+1} - sa_i - b_{i+1} + b_i)} .$$

A similar expression exists for the wedge in the (s,t)-plane above ℓ_i and below ℓ_{i+1}.

We observe that inside the double wedge defined by ℓ_i and ℓ_{i+1}, μ_i is a fraction that has the unknown s in the denominator. To minimize μ, we must solve

$$\min_{s,t} \sum_{i=1}^{n-1} \mu_i(s,t) ,$$

which is a sum of fractions and involves solving a polynomial of linear degree, if the minimum occurs where many edges of f and g intersect. It appears that we cannot hope to get an exact solution in this case.

Approximation for μ. We continue to derive an approximation algorithm. For any given $\varepsilon > 0$, we will determine values of s and t such that the value of μ is at most $(1+\varepsilon)$ times μ^*, the minimum possible value of μ. Our approach is to linearize each function μ_i, so that the sum we get is linear as well.

We first analyze the function μ_i closer. For a fixed scaling factor $\hat{s}$, consider the function $\mu_i(\hat{s},t)$. From Lemma 2 we know that below ℓ_i and ℓ_{i+1}, or above both of them, $\mu_i(\hat{s},t)$ is linear in t with slope $x_i - x_{i+1}$ and $x_{i+1} - x_i$. Between ℓ_i and ℓ_{i+1}, $\mu_i(\hat{s},t)$ is a quadratic function in t. In fact, $\mu_i(\hat{s},t)$ is symmetric, differentiable everywhere, and consists of three pieces, see Figure 2. Since this holds for any scaling factor $\hat{s}$ except the one of the apex of the double wedge (where it has two pieces and is not differentiable in the apex), we obtain useful properties of the whole function μ_i.

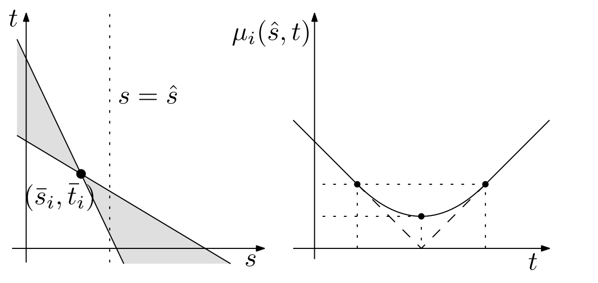


Figure 2: A double wedge in the s, t-plane, and a vertical cross-section of it that shows $\mu_i(\hat{s}, t)$.

Lemma 3 *μ_i is a convex function, and the restriction of μ_i to a ray starting at $(\bar{s}_i, \bar{t}_i)$ is a linear function.*

Corollary 4 *μ is a convex function.*

To obtain a $(1+\varepsilon)$-approximation, we will replace μ_i by a piecewise linear function. Since μ_i is already linear outside the double wedge, we only need to do so inside it. We do so by first approximating the quadratic part of $\mu_i(\hat{s}, t)$. We choose $\lceil 2/\sqrt{\varepsilon} \rceil$ extra points equally-spaced on $\mu_i(\hat{s}, t)$ such that the maximum distance between the polygonal line through the chosen points and $\mu_i(\hat{s}, t)$ itself is at most $\varepsilon\mu_i(\hat{s}, t)$, see Figure 3. We can extend this 1-dimensional approximation to a 2-dimensional one by choosing lines through the new points and $(\bar{s}_i, \bar{t}_i)$, giving a partition of the plane into $\Theta(1/\sqrt{\varepsilon})$ wedges. Lemma 3 ensures that the implied piecewise linear function—denoted $\tilde{\mu}_i$—is a $(1+\varepsilon)$-approximation of μ_i everywhere.

Now we are ready to consider all edges from f and g. If we were considering the exact function μ, we would get $n-1$ double wedges in the (s, t)-plane. They form an arrangement of $O(n^2)$ cells, and in each cell we can write down a closed expression for μ as the sum of the μ_i. However, this gives us the expression that we cannot optimize exactly. Using the piecewise linear approximations $\tilde{\mu}_i$, we get an arrangement of $O(n/\sqrt{\varepsilon})$ lines which partition the (s, t)-plane in $O(n^2/\varepsilon)$ cells. The sum of at most $n-1$ linear functions is of course linear, and with little extra work we can easily find the minimum of $\tilde{\mu}(s, t)$ in $O(n^2/\varepsilon)$ time.

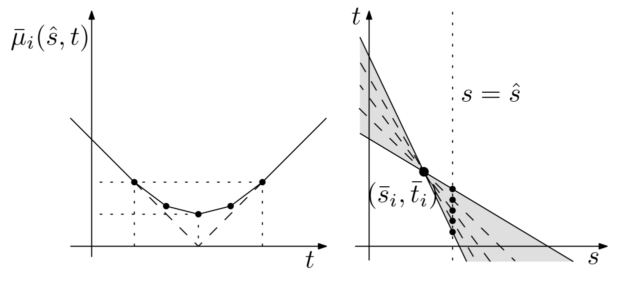


Figure 3: The approximation $\bar{\mu}_i(\hat{s}, t)$ and the subdivision into double wedges in the s, t-plane.

A more efficient solution is obtained using cuttings for a divide-and-conquer approach. This works because all $\tilde{\mu}_i$ are convex functions, so $\tilde{\mu}$ is convex as well. Let $n' = n/\sqrt{\varepsilon}$, and consider the $O(n')$ lines that define the boundaries of the cells over which $\tilde{\mu}$ is linear. We compute a $(1/2)$-cutting of size $O(1)$ in $O(n')$ time [2]. On each of the $O(1)$ edges in the cutting, we find the minimum in $O(n')$ time, which will reveal in which triangle of the cutting the global optimum of $\tilde{\mu}$ lies. We recurse on the lines that intersect this triangle; their number is halved by the definition of $(1/2)$-cuttings.

After $O(\log n')$ phases of divide-and-conquer, which together take $O(n')$ time, we find the triangle that lies in a single cell in the arrangement of all $O(n')$ lines that contains the global optimum.

Theorem 5 *Let f and g be two one-dimensional terrains with n vertices. For any fixed $\varepsilon > 0$, a linear transformation of the image of f that yields at most area $(1+\varepsilon)A^*$ between the images of g and the transformed f can be determined in $O(n/\sqrt{\varepsilon})$ time, where A^* is the minimum area achievable.*

4 Conclusions and open problems

The obvious open problem is extending the result on one-dimensional terrains to two-dimensional terrains, which is what we are still working on. Another extension we consider is a third measure for matching, namely the integral of squared difference between the functions.

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[2] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993.

[3] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

[4] K. L. Clarkson. Linear programming in $O(n3^{d^2})$ time. *Inf. Process. Lett.*, 22(1):21–24, 1986.

[5] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995.

[6] M. de Berg, H. J. Haverkort, S. Thite, and L. Toma. I/O-efficient map overlay and point location in low-density subdivisions. In *ISAAC*, volume 4835 of *Lecture Notes in Computer Science*, pages 500–511, 2007.

Cutting an Organic Surface

Jean-Marie Favreau[*], Vincent Barra[†]

Abstract

Mapping a given surface onto a piece of the plane is an usual problem, and conformal mappings with border free approaches are classical as non-distorting methods. However, algorithms of this category are only able to produce a mapping from a surface topologically equivalent to a disc. In the general case, a topological cutting is thus required.

The approach we are reporting here is a method that does not only take into account the topological properties by cutting the original surface into a disc [3], but also the the geometrical properties by globally selecting the local extrema of original arm-like surfaces (denoted organic surfaces in the following).

1 Introduction

Various approaches has been proposed to map a surface onto a piece of the plane. Sheffer *et al.* [6] described a global framework that use a topological cutting step and a geometrical detection of local extrema. These points are selected using an estimation of the curvature on a neighbourhood. The topological step neither care about the globally shortest loops, nor the full geometrical properties of the surface. Tierny *et al.* [8] detected some extrema points during their Reeb Graph construction. After selecting the most distant pair of points v_1 and v_2, a geodesic distance from each of these points is computed. The intersection of the extrema for each distance is computed, determining the local maxima of the whole surface. This method seems to be restricted by surfacic noise, and by the approximation step of the intersection. Gu *et al.* [4] used a global approach to detect the extrema by using a fixed border parameterization of the surface, then selecting the triangle with maximum geometric stretch as the extremum of a path from the existing boundary. The parameterization and cutting step are applied until the stretch's stabilisation of the parameterization. Beyond the multiple parameterization steps, the main limitation of this method is the use of a fixed border parameterization. Positionning the boundary points on the plane is thus not easy, and the positionning is not unique. Moreover, the stretch of the triangles is largely dependant on the boundary shape and distribution.

We propose here a method that first uses a topological cutting step, handling the geodesic distances, and can be influenced by some other informations like visibility. Then we perform a single free border parameterization to select the extrema according to the global geometry of the surface.

In the following, we will detail this approach, first describing the topological cutting step that takes into account the geodesic distances and can be influenced by some other information like visibility. Then we will introduce the use of a free border parameterization to select the extrema, and the final cutting process to provide a geometrical and topological cutting of the original surface onto a disc. Next some variations on the distance method will be detailed, and some experimental results will finally be presented.

2 Background

In the following, surfaces are described by 2-meshes, that contain both topological information and a practical description (as a triangulated surface):

Definition 1 *A* 2*-mesh* $\mathcal{M}$ *is a 2-manifold simplicial 2-complex (with or without boundaries), i.e.* $\mathcal{M}$ *is the union of 0-simplices called vertices, 1-simplices called edges and 2-simplices called triangles, and locally homeomorphic*[1] *to either a disc or an half disc.*

To describe a cutting on a 2-mesh, 1-paths and 1-loops are used:

Definition 2 *A* 1-path *of a* 2*-mesh* $\mathcal{M}$ *is a list of* n *oriented edges* $s_0, ..., s_{n-1}$ *when for each* $i = \{0, ..., n-2\}$*, the terminal vertex of* s_i *is the initial vertex of* s_{i+1}*. A* 1-loop *of a* 2*-mesh* $\mathcal{M}$ *is a* 1*-path* $s_0, ..., s_{n-1}$ *where the initial vertex of* s_0 *is the terminal vertex of* s_{n-1}*.*

A cutting along a 1*-path* p is naturally performed by duplicating vertices and edges of p, thus removing junctions between triangles adjacent to p.

Definition 3 *A non-separating* 1*-loop in* $\mathcal{M}$ *is a* 1*-loop with the property that* $\mathcal{M}$ *can be cut ac-*

[*]Université Blaise Pascal / LIMOS UMR CNRS 6158 Jean-Marie.Favreau@isima.fr

[†]Université Blaise Pascal / LIMOS UMR CNRS 6158 Vincent.Barra@isima.fr

[1]Two surfaces A and B are homeomorphic if there is a bi-continuous function from A to B.

cording to this 1-loop without separating $\mathcal{M}$ into more than one connected component.

3 Selecting the extrema

The selection of extrema on a 2-mesh $\mathcal{M}$ can be globally applied by studying the stretch of edges on a parameterization. In fact, a conformal parameterization like ABF [7] produces a mapping between a surface homeomorphic to a disc and a part of the plane. The stretch of the edges is a direct and global consequence of the geometrical properties of the original surface, *e.g.* each leg of the horse is clustered on a small part of the plane (Figure 1(b)).

3.1 Topological cutting

A cutting preprocess is essential to produce the mapping where extrema will be detected. In fact, parameterization methods are mainly able to build the mappings from a surface homeomorphic to a disc. Such a preprocess should compute a cutting:

Definition 4 *A cutting C of a 2-mesh $\mathcal{M}$ is a set of edges whose complement in $\mathcal{M}$ is homeomorphic to a disc.*

In a surface homeomorphic to a disc, an empty set of edges is a cutting of the surface, but for more topologically complex surfaces, a mandatory cutting step is required.

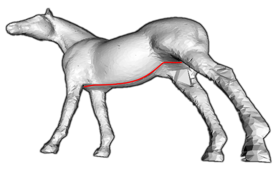
3.1.1 Non trivial surfaces

Erickson and Har-Peled [3] showed that computing the shortest cutting of a given surface is NP-hard in general. We used here a variation of their algorithm to compute an approximation of this cutting:

```
Data: a 2-mesh $\mathcal{M}$ non homeomorphic to a sphere
while the genus of $\mathcal{M}$ is not 0 do
    Find $c$ the shortest non-separating 1-loop of $\mathcal{M}$;
    Cut $\mathcal{M}$ according to $c$;
while the number of boundaries of $\mathcal{M}$ is not 1 do
    Find $c$ the shortest 1-path of $\mathcal{M}$ between two different boundaries;
    Cut $\mathcal{M}$ according to $c$;
```

Algorithm 1: Cutting a surface into a disc

Computing the shortest 1-path and 1-loop uses the Dijkstra algorithm [1], using the 3D length $l_3(v_1, v_2)$ of each edge (v_1, v_2) as a cost function. The shortest non-separating 1-loop is computed by first building a cutting by an immediate method, then from each point b of the cutting by looking to the shortest non-separating 1-loop starting from b, using a growing surface.


(a) The original surface (homeomorphic to a sphere, 16424 triangles) and the topological cutting (in red).

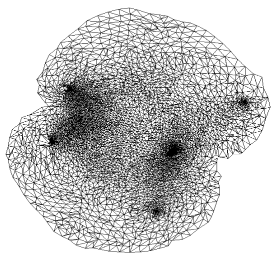

(b) The unfolded surface using ABF [7].

Figure 1: A topological cutting and the associated conformal mapping.

However this method cannot be applied on a surface homeomorphic to a sphere: there is no non-separating 1-loop on such a surface.

3.1.2 Surfaces homeomorphic to a sphere

Given a 2-mesh homeomorphic to a sphere, a non-empty 1-path (but not 1-loop) is a valid cutting. To facilitate the parameterization step, we performed this cutting using Algorithm 2, using first Principal Components Analysis (PCA) [2] to detect the direction of the main axis from the center of mass.

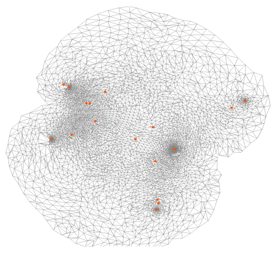
```
Data: a 2-mesh $\mathcal{M}$ homeomorphic to a sphere
Compute the main axis $A$ using PCA;
Build the list of triangles of $\mathcal{M}$ intersecting $A$;
Order the list according to the position on $A$;
Select a point in the first and last triangle;
Build the cutting 1-path using the Dijkstra algorithm between the two selected points;
```

Algorithm 2: Cutting a sphere into a disc

PCA has been first applied on the points of the 2-mesh. In a regular mesh, this approach never fails to detect the main axis and produce a cutting allowing an efficient parameterization step (Figure 1(a)). But for surfaces with non regular triangles, incoherent cuttings were observed. Rather than remeshing the surface, a volumic approach may be employed, by voxelizing the object described by the surface [5], then computing PCA axis using voxels inside the object.

3.2 Distortion

Given a 2-mesh homeomorphic to a disc, a conformal parameterization can be applied [7], producing a locally one-to-one mapping with a piece of the plane. As illustrated on Figure 1, this mapping contains the geometrical information we are looking for: in this figure, legs and the head (extrema) are projected by the parameterization on high density areas. Because of the conceiv-


(a) Extrema computed using the Dijkstra Algorithm from the boundary vertices, using $D(\cdot,\cdot)$ as the cost function (23 extrema)

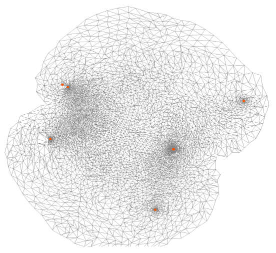
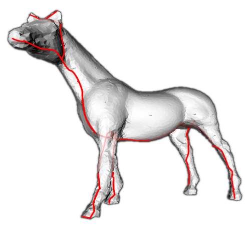

(b) Selected extrema by using the median of the distortion (12 extrema)

Figure 2: Computation of the extrema using the distortion $D(\cdot,\cdot)$ as edges' distance on a parameterization (Figure 1).


(a) Cutting of the original surface (Figure 1(a)) using Algorithm 3 with the selected extrema shown in Figure 2(b).

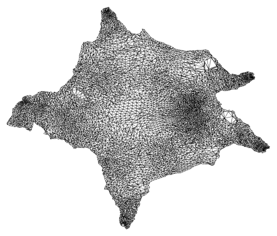

(b) The unfolded surface using ABF.

Figure 3: A geometrical cutting and the associated conformal mapping.

able variation of the triangle's size on the original mesh, the density of the areas in the plane is not the real information that has to be taken into account. The distortion of the original surface induced by the parameterization rather provides a more informative parameter.

Let $l_3(v_1, v_2)$ be the length of the edge (v_1, v_2) on the original 2-mesh, and $l_2(v_1, v_2)$ the length of the edge on the plane. $D(v_1, v_2) = l_3(v_1, v_2)\,(l_2(v_1, v_2))^{-1}$ describes for each edge the distortion produced by the parameterization. By applying the Dijkstra algorithm from the boundary vertices of the 2-mesh, a distance $L_b(v)$ from the boundary for each vertex v is computed. $D(\cdot,\cdot)$, used as the cost function, produces a partial order that takes into account the geometry of the 2-mesh described by the distortion produced by the parameterization.

The local maxima $(m_i)_{0\leq i<n}$ of D are in particular located on high density areas (Figure 2(a)). To select the most pertinent vertices, we first ordered $(m_i)_{0\leq i<n}$ according to the distortion of each vertex v computed by $D(v) = \sum_{(v,v_2)\in E} D(v, v_2)$, where E is the set of the edges of $\mathcal{M}$. The selected extrema are finally chosen using a threshold. For automatic selection, we used the median of the distortion of the vertices (Figure 2(b)). On a user interface, this threshold should be easily choosen using a cursor on the ordered list of selected extrema.

4 Cutting the surface

The topological cutting described in section 3.1 produces a surface homeomorphic to a disc, using topological information. Using the extrema selected in section 3.2, we can now modify this cutting by introducing some geometrical information. However, the final surface also has to be homeomorphic to a disc.

```
Data: a 2-mesh $\mathcal{M}$ non homeomorphic to a
      disc and a set $V$ of vertices of $\mathcal{M}$ ;
Sort $V$ according to the geodesic distance
from the nearest of the boundary of $\mathcal{M}$ to the
furthest;
foreach $v \in V$ do
    Find $c$ the shortest 1-path of $\mathcal{M}$ between
    $v$ and the boundary of $\mathcal{M}$;
    Cut $\mathcal{M}$ according to $c$;
```

Algorithm 3: Cutting a surface into a disc

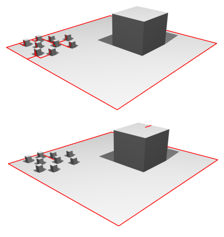
Each step of the Algorithm 3 enlarges the boundary without modifying the topology of the surface, introduces the minimal cutting to add the selected vertex to the boundary, and produces a cutting that takes into account the geometry of the original surface (Figure 3(a)).

5 Variations on distance

The computations described in the previous sections are based on the length $l_3(v_1, v_2)$ of the edges (v_1, v_2) of the 2-mesh in $\mathbb{R}^3$. A basic approach is to use the Euclidean distance $|v_1, v_2|$ as the length of the edges. Sheffer *et al.* [6] used a multiplier coefficient corresponding to the visibility of the edge.

This Eudidean distance can also be modified to take into account the local curvature, encouraging the paths to go through the acute edges. Let $\alpha(v_1, v_2)$ be the angle defined by the normals of the adjacents triangles. A multiplier coefficient can be defined by $(1 - c^{\frac{|\alpha(v_1,v_2)|}{\pi}})$, $c \in [0, 1]$.

Finally, in context of texture mapping, a painting tool should be offered to the graphist to manually describe the coefficient. Our method is then not limited as an automatic cutting, but can be driven by the user.


(a)

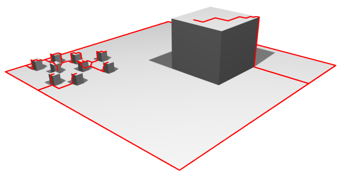

(b)

Figure 4: Cuttings of a surface with various scales of details. Left: using [8] to detect the extrema (top: with a small ϵ; bottom: with a larger value of ϵ). Right: using our method.

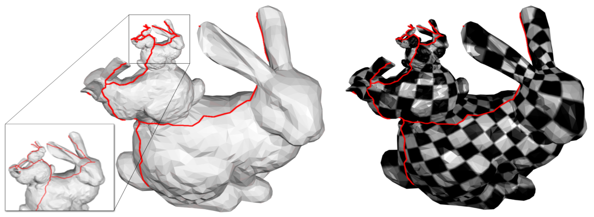


Figure 5: A geometrical cutting of a surface (6,789 points and 13,510 triangles), with multiscale details.

6 Results

A `C++` implementation of these cutting algorithms has been performed to illustrate results of our approach. For the distortion step, we used an implementation of [7], but a linear approach [9] can also be used.

One of the main defaults of global approaches of local extrema detection is the inability to extract extrema from various scale of details. Thus [8] used a ϵ value for matching the extrema. Figure 4(a) illustrates the limitation of this approach, that cannot take into account small and big details at the same time. On the contrary, our approach (Figure 4(b)) is able to detect at the same time and without adjustment multiscale details. Figure 5 is an illustration of the detection of multiscaling details.

Table 1 compiles the computation times of our method for surfaces with variable genus, boundary number and size (number of triangles). Each of those variables modifies the duration of the algorithm, but it seems to be tolerable for off-line applications.

7 Conclusion

We presented here a global approach to cut a surface taking into account the geometrical properties of organic surfaces. The resulting cut surface thus can be unfold using a conformal mapping with minor surfacic distortions (Fig. 5 right), thanks to the topological and geometrical approach.

Surface	sect. 3.1	Unfolding	sect. 4
3-T	36.71 s	7.96 s	15.78 s
Rabbits	0.33 s	7.14 s	1.35 s
Horse	0 s	5.41 s	1.89 s

Table 1: The three steps of our algorithm applied on various surfaces (2.6 GHz CPU). 3-T: a 3-genus surface of 55,296 triangles, without boundary. Rabbits (Fig. 5): a 0-genus surface of 13,510 triangles with 5 boundaries. Horse: a 0-genus surface of 16,424 triangles without boundary.

References

[1] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[2] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[3] J. Erickson and S. Har-Peled. Optimally Cutting a Surface into a Disk. *Discrete & Computational Geometry*, 31(1):37–59, 2004.

[4] X. Gu, S. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

[5] D. Haumont and N. Warzée. Complete polygonal scene voxelization. *Journal of Graphics Tools*, 7(3):27–41, 2002.

[6] A. Sheffer and J. C. Hart. Seamster: inconspicuous low-distortion texture seam layout. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 291–298, Washington, DC, USA, 2002. IEEE Computer Society.

[7] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogom yakov. Abf++ : Fast and robust angle based flattening. *ACM Transactions on Graphics*, Apr 2005.

[8] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Invariant High-Level Reeb Graphs of 3D Polygonal Meshes. In *3rd IEEE International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'06)*, pages 105–112, Chapel Hill, North Carolina, USA, 2006.

[9] R. Zayer, B. Lévy, and H.-P. Seidel. Linear angle based parameterization. In *ACM/EG Symposium on Geometry Processing conference proceedings*, 2007.

Two-Convex Polygons[*]

O. Aichholzer[†] F. Aurenhammer[‡] F. Hurtado[§] P.A. Ramos[¶] J. Urrutia[‖]

Abstract

We introduce a notion of k-convexity and explore some properties of polygons that have this property. In particular, 2-convex polygons can be recognized in $O(n \log n)$ time, and k-convex polygons can be triangulated in $O(kn)$ time.

1 Introduction

The notion of convexity is central in geometry. As such, it has been generalized in many ways and for different reasons. In this note, we consider a simple and intuitive generalization, which to the best of our knowledge has not been worked on. It leads to an appealing class of polygons in the plane, with interesting structural and algorithmic properties.

A set in $\mathbb{R}^d$ is convex if its intersection with every straight line is connected or empty. This definition may be relaxed to directional convexity or *D-convexity* [9, 14], by considering only lines parallel to one out of a (possibly infinite) set D of vectors. A special case is *ortho-convexity* [16], where only horizontal and vertical lines are allowed. For any fixed D, the family of D-convex sets is closed under intersection, and thus can be treated in a systematic way using the notion of *semi-convex* spaces [17], which is sometimes appropriate for investigating visibility issues. The *D-convex hull* of a set M is the intersection of all D-convex sets that contain M. If D is a finite set, this definition of a convex hull may lead to an undesirably sparse structure—an effect which can be remedied by using a stronger, functional (rather than set-theoretic) concept of D-convexity [14].

k-Convex Sets We consider a different generalization of convexity. A set M in $\mathbb{R}^d$ is called *k-convex* if

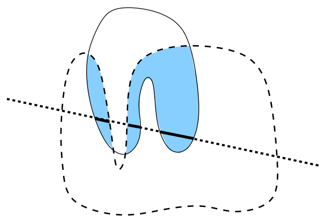


Figure 1: Intersecting two 2-convex sets

there exists no straight line that intersects M in more than k connected components. Note that 1-convexity refers to convexity in its standard meaning[1]. To reformulate in terms of visibility, call two points $x, y \in M$ to be *k-visible* if $\overline{xy} \cap M$ consists of at most k components. Thus, a set is k-convex if and only if any two of its points are mutually k-visible. Applications of this concept may stem from placement problems for modems that have the capacity of sending through a fixed number of walls. Unlike directional convexity, k-convexity fails to show the intersection property: The intersection of k-convex sets is not k-convex, in general (k fixed), cf. Figure 1. For $k \geq 2$, a k-convex set M may be disconnected, or if being connected, its boundary may be disconnected. In this note, we will restrict attention to simply connected sets in two dimension, namely, simple polygons in the plane.

There are two notions of planar convexity that appear to be close to ours. One is *k-point convexity* [18, 4] which requires that, for any k points in a set M in $\mathbb{R}^2$, at least one of the line segments they span is contained in M. (Thus 2-point convex sets are precisely the convex sets.) The other is *k-link convexity* [13], being fulfilled for a given polygon P if, for any two points in P, the geodesic path connecting them inside P consists of at most k edges. (The 1-link convex polygons are just the convex polygons.) While there is a relation between k-convexity and the former concept, the latter concept is totally unrelated.

In the following we study basic properties of k-convex polygons (Section 2), give a characterization of 2-convex polygons (Section 3), and present efficient algorithms for recognizing (Section 4) and triangulating (Section 5) such polygons. Finally, Section 6 offers a discussion of our results.

[*]Partially supported by the FWF Joint Research Program 'Industrial Geometry' S9205-N12, Projects MEC MTM2006-01267, DURSI 2005SGR00692, MEC MTM2008-04699-C03-02, and the bilateral Spain-Austria program 'Acciones Integradas' ES 01/2008.

[†]Institute for Software Technology, University of Technology, Graz, Austria, oaich@ist.tugraz.at

[‡]Institute for Theoretical Computer Science, University of Technology, Graz, Austria, auren@igi.tugraz.at

[§]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain, Ferran.Hurtado@upc.edu

[¶]Departamento de Matemáticas, Universidad de Alcalá, Madrid, Spain, pedro.ramos@uah.es

[‖]Instituto de Matemáticas, Universidad Nacional Autònoma de Mèxico, urrutia@matem.unam.mx

[1]We face notational ambiguity. The term 'k-convex' has, maybe not surprisingly, been used in different settings, namely, for functions [15], for graphs [3], and for discrete point sets [11]. Also, the concept of k-point convexity [18] has later been called k-convexity in [4].

2 Two-Convex Polygons

From now on, all geometric objects we will consider are closed sets in the Euclidean plane. Let P be a simple polygon, and denote with n the number of edges of P. Two line segments e and e' are said to *cross* if $e \cap e'$ is a point in the relative interior of both e and e'. Clearly, a polygon P is k-convex if every line segment with endpoints in P crosses at most $2(k-1)$ edges of P. The *stabbing number* [8] of a set of (interior-disjoint) line segments is the largest number of crossings attainable with a straight line. A polygon is k-convex if and only if its stabbing number is at most $2k$. Thus, all our observations on k-convexity could be reformulated in terms of stabbing numbers.

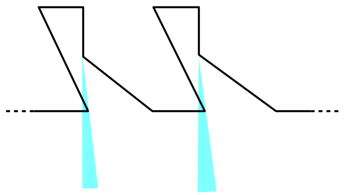


Figure 2: Quadratic 2-kernel construction

To see that 2-convexity is already significantly more complex than standard convexity, consider the *k-kernel* of a polygon P, i.e., the subset $M_k \subset P$ such that the entire polygon P is k-visible from each point $x \in M_k$. Note that P is k-convex if and only if $P = M_k$. Whereas M_1 is known to be a convex set which is computable in $O(n)$ time [12], M_2 may have an $\Omega(n^2)$ description; see Figure 2. The shaded areas emanating from the spikes are not part of M_2, and arranging such spikes along the boundary of a rectangle leads to a grid-like structure that partitions the 2-kernel into a quadratic number of components.

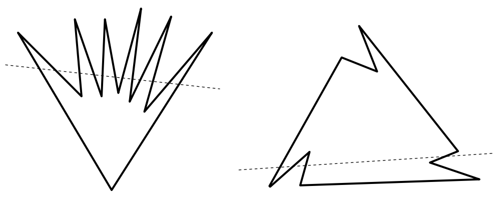


Figure 3: Star-shaped versus 2-convex

There is also no immediate relation to *star-shaped* polygons, i.e., polygons P with $M_1 \neq \emptyset$. Figure 3 shows, on the lefthand side, a polygon which is star-shaped but only $\frac{n}{2}$-convex. On the righthand side, we see a polygon which is 2-convex but not star-shaped. Visually, 2-convexity seems to be closer to convexity than is star-shapedness.

While 2-convexity clearly restricts the winding number [19] of a polygon, its link distance [13] is unaffected and may well be $\Theta(n)$. Conversely, a polygon which is 2-link convex (such that any two of its points are at link distance 2 or less) may fail to be k-convex for sublinear k. The star-shaped polygon in Figure 3 (left) is an example.

There is, interestingly, a relation to *k-point* convexity as defined in [18]. Every k-point convex polygon P is $(k-1)$-convex. To verify this, assume the contrary, which implies the existence of a straight line L which intersects P in at least k components. Select a point in each component. Now, by the assumed k-point convexity of P, at least one pair among the selected points yields a line segment, S, which entirely lies in P. As, clearly, $S \subset L$, the corresponding two components cannot be different—a contradiction. No implication exists in the other direction, however. For example, the 2-convex polygon in Figure 3 (right) fails to be 3-point convex. The class of k-convex polygons also differs from the class of k-guardable polygons defined in [1]. A linear number of (point) guards may be needed already to watch a 2-convex polygon. For further details see the full version of this paper.

3 Characterization

The definition of a k-convex polygon does not translate into an algorithm for recognizing such polygons. We give a characterization of k-convex polygons that allows a decision in time $O(n \log n)$.

Let P be the polygon under consideration, and denote with ∂P its boundary. A line L is called a *j-stabber* of P if L crosses ∂P at least j times. Note that a j-stabber may totally contain edges of P; these are *not* considered to contribute to the count. An *inflection edge* of P is an edge between a convex and a reflex vertex of P. Finally, an *inner tangent* of P is a line segment $T \subset P$ such that T contains two non-adjacent reflex vertices of P in its relative interior.

Lemma 1 *A simple polygon P is 2-convex if and only if P has no inner tangent, and no 3-stabber that contains an inflection edge.*

Proof. Omitted. □

4 Recognition

Let us assume that the given polygon P is nonconvex, as things are trivial, otherwise. The recognition algorithm is based on Lemma 1. It looks for inner tangents and 3-stabbers at inflection edges, trying to witness that P is not 2-convex. To this end, ray shooting [5] is performed for each reflex vertex of P, in the directions of its two incident edges. If ∂P is intersected more than once in a fixed direction, then a 6-stabber exists and we report that P is not 2-convex and stop. This covers the necessary check at each inflection edge.

The algorithm continues if all these directions yield a *unique* intersection point with ∂P. Let us assume, for the remainder of this section, that P is of this form. We store the points of intersection, and use them to check for inner tangents. Define, for each reflex vertex v of P, its *critical range* $C(v)$ as the set of all points $x \in \partial P$ such that $\overline{xv}$ can be prolonged to a line segment being tangent to P at v. Note that such a segment need not lie entirely in P. However, $C(v)$ consists of exactly two connected intervals on ∂P, whose endpoints are among the stored points obtained from ray shooting. See Figure 4, where $C(v)$ is drawn with bold lines.

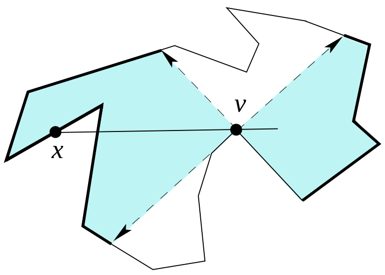


Figure 4: Critical range for vertex v

Observation 1 *P admits an inner tangent if and only if P has two reflex vertices v and v' such that $v \in C(v')$ and $v' \in C(v)$.*

Proof. Omitted. □

The strategy for detecting inner tangents is now clear. We first augment each reflex vertex with the two intervals of polygon vertices that lie inside its critical range. Then, we scan around ∂P with a point x and maintain, in some search tree, the set $R(x)$ of reflex vertices whose critical ranges contain x. That is, after initialization for a fixed position of x, we update $R(x)$ whenever x scans over some hitting point from ray shooting. Moreover, when x reaches some reflex vertex v of P, we search the tree with the four vertices u_i, u_j, u_k, u_ℓ that delimit $C(v)$, to check for $R(x) \cap [u_i, u_j] = \emptyset$ and $R(x) \cap [u_k, u_\ell] = \emptyset$.

The number of events where $R(x)$ undergoes some change or is searched is $O(n)$. This gives $O(n \log n)$ time, as does the total time spent for the $O(n)$ initial ray shooting queries; see [5]. The space requirement remains in $O(n)$.

Theorem 2 *For a simple polygon with n vertices, 2-convexity can be decided in $O(n \log n)$ time and $O(n)$ space.*

5 Triangulation

Triangulating a polygon in $O(n)$ time with a reasonably simple algorithm is still outstanding, except for special classes of polygons described, e.g., in [7, 1]. We will show below that 2-convex polygons add to this list. A simple ear-cutting-type triangulation method can be used, based on the following property.

Observation 2 *If P is a 2-convex polygon then, for each reflex vertex v of P, its critical range $C(v)$ is visible from v.*

Proof. Let P be 2-convex. Consult Figure 4 again, and consider any point $x \in C(v)$. The line segment $\overline{xv}$ does not cross ∂P because, otherwise, $\overline{xv}$ could be prolonged and slightly translated to yield a 6-stabber of P. □

```
Algorithm CUT-TO-PIECES
  $v_0 \leftarrow$ reflex vertex of $P$
  $v \leftarrow v_0$
  repeat
    Triangulate from $v$ to $C(v)$
    $v \leftarrow$ next reflex vertex along $\partial P$
  until $v = v_0$
```

Triangulating from a given vertex refers to the yet untriangulated part of the polygon P. Figure 5 illustrates the effect of visiting the reflex vertices of P in clockwise order. After the loop, each subpolygon Q left untriangulated has a special property: Each vertex w of Q sees *all* vertices of Q in its internal angle (not just those in its critical range if w is reflex). Assuming the contrary implies that w is endpoint of some line segment tangent to the original polygon P at a reflex vertex, say v. But then we have $w \in C(v)$, and Q would have been split with the edge $\overline{vw}$ by Algorithm CUT-TO-PIECES. Observe that this argumentation does *not* imply that left-over polygons are star-shaped. Still, we can easily complete the triangulation for P by adding diagonals for such polygons.

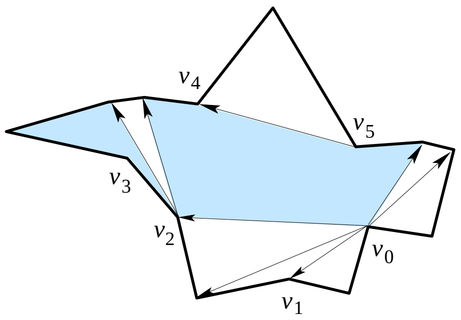


Figure 5: Ear cutting leaves two subpolygons

An alternative fast triangulation method for general k-convex polygons results from the fact that we can sort the vertices of a k-convex polygon P in any given direction (say, x-direction) in $O(kn)$ time: Simply scan around ∂P and use insertion sort, starting each time from the place where the x-value of the previous vertex has been inserted. Then any fixed value x_j, once being inserted, takes part in later comparisons at most $2k - 1$ times because, otherwise, the vertical line $x = x_j$ would intersect P in more than k components. Having x-sorted P's vertices, a simplified plane sweep method can be used to build a vertical trapezoidation [6, 10] (and then a triangulation) of P. Only trivial data structures are needed, as the scenario on the sweep line is of complexity $O(k)$, by the k-convexity of P. Thus, each vertex of P can be processed in $O(k)$ time during the sweep. An $O(kn)$ time triangulation algorithm results.

Theorem 3 *Any k-convex polygon can be triangulated in $O(kn)$ time and $O(n)$ space.*

6 Discussion

Among the open algorithmic problems raised by this paper is the recognition of 2-convex polygons in linear time. In general, for $k \geq 3$, recognizing k-convexity of a polygon in subquadratic time is open. Also, no computational discussion of k-point convexity apparently exists.

As a combinatorial question, is it always possible to build, on top of a given planar point set, a 2-convex decomposition with a sublinear number of polygons? The problem of constructing a polygonization (a polygonal cycle through the points) which has k-convexity as low as possible seems to be hard. Is there a relation to the reflexivity [2] of point sets? How fast can we decide whether a point set admits a 2-convex polygonization?

Let us finally show that there are point sets where the best polygonization is at least $\Omega(\sqrt{n})$-convex. To this end, let S be the n points of a $\sqrt{n} \times \sqrt{n}$ grid, slightly perturbed to be in general position. Let L be a set of $\sqrt{n} - 1$ horizontal and $\sqrt{n} - 1$ vertical lines which can be drawn between the different rows and columns to separate the grid points. Then any edge of an arbitrary polygonization P of S intersects at least one element of L. Assign each edge of P to one of the elements in L it intersects. This way on average each line in L gets assigned $\frac{n}{2\sqrt{n}-2}$ edges of P. Thus, by the pigeon-hole principle, there is at least one line in L which intersects $\Omega(\sqrt{n})$ edges of P, that is, P is at least $\Omega(\sqrt{n})$-convex. We close with the question whether we can always find a polygonization which is $o(n)$-convex.

Acknowledgments

We would like to thank Clemens Huemer, David Rappaport, and Vera Sacristán for fruitful discussions.

References

[1] G. Aloupis, P. Bose, V. Dujmovic, C. Gray, S. Langerman, B. Speckmann. *Triangulating and guarding realistic polygons.* Proc. 20th Canadian Conference on Computational Geometry, 2008.

[2] E.M. Arkin, S.P. Fekete, F. Hurtado, J.S.B. Mitchell, M. Noy, V. Sacristán, S. Sethia. *On the reflexivity of point sets.* In: B. Aronov, S. Basu, J. Pach, M. Sharir (eds.) Discrete & Computational Geometry: The Goodman-Pollack Festschrift, 2003, 139-156.

[3] D. Artigas, M.C. Dourado, J.L. Szwarcfiter. *Convex partitions of graphs.* Electronic Notes in Discrete Mathematics 29 (2007), 147-151.

[4] M. Breen, D.C. Kay. *General decomposition theorems for m-convex sets in the plane.* Israel Journal of Mathematics 24 (1976), 217-233.

[5] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, J. Snoeyink. *Ray shooting in polygons using geodesic triangulations.* Algorithmica 12 (1994), 54-68.

[6] B. Chazelle, J. Incerpi. *Triangulation and shape-complexity.* ACM Transactions on Graphics 3 (1984), 135-152.

[7] H. ElGindy, G.T. Toussaint. *On geodesic properties of polygons relevant to linear time triangulation.* The Visual Computer 5 (1989), 68-74.

[8] S.P. Fekete, M.E. Lübbecke, H. Meijer. *Minimizing the stabbing number of matchings, trees, and triangulations.* Proc. 15th Ann. ACM-SIAM Symp. on Discrete Algorithms, 2004, 430-439.

[9] E. Fink, D. Wood. *Fundamentals of restricted-orientation convexity.* Information Sciences 92 (1996), 175-196.

[10] A. Fournier, D.Y. Montuno. *Triangulating simple polygons and equivalent problems.* ACM Transactions on Graphics 3 (1984), 153-174.

[11] G. Kun, G. Lippner. *Large empty convex polygons in k-convex sets.* Periodica Mathematica Hungarica 46 (2003), 81-88.

[12] D.T. Lee, F.P. Preparata. *An optimal algorithm for finding the kernel of a polygon.* Journal of the ACM 26 (1979), 415-421.

[13] A. Maheshwari, J.-R. Sack, H. Djidjev. *Link distance problems.* In: Handbook of Computational Geometry, J.-R. Sack and J. Urrutia (eds.), Elsevier, 2000, 519-558.

[14] J. Matoušec, P. Plecháč. *On functional separately convex hulls.* Discrete & Computational Geometry 19 (1998), 105-130.

[15] T. Pennanen. *Graph-convex mappings and K-convex functions.* Journal of Convex Analysis 6 (1999), 235-266.

[16] G.J.E. Rawlins, D. Wood. *Ortho-convexity and its generalizations.* In: Computational Morphology, G.T. Toussaint (ed.), North-Holland Publishing Co., Amsterdam, 1988, 137-152.

[17] S. Schuierer, D. Wood. *Visibility in semi-convex spaces.* Journal of Geometry 60 (1997), 160-187.

[18] F.A. Valentine. *A three-point convexity property.* Pacific Journal of Mathematics 7 (1957), 1227-1235.

[19] G. Vegter. *Kink-free deformations of polygons.* Proc. 5th Ann. ACM Symp. Computational Geometry, 1989, 61-68.

Geometric Measures on Imprecise Points in Higher Dimensions[*]

Hein Kruger[†] Maarten Löffler[†]

Abstract

Imprecision in input data is an important obstacle to the application of techiniques from computational geometry in practical situations. Recently, a growing amount of research acknowledges this issue. However, the focus has mostly been on the planar case, while many applications work on data in three or more dimensions. In particular, van Kreveld and Löffler [10] study the problem of computing lower and upper bounds on the possible values of several basic geometric measures on point sets in $\mathbb{R}^2$. In this work, we investigate how these results generalise to higher dimensions. Interestingly, some of the algorithms and hardness results go through almost unchanged, while in other cases some distinctly different ideas are needed.

We present results for two extent measures on point sets: the axis-aligned minimum bounding box and the width. The results range from algorithms that run in linear time in any dimensions, to running times of $n^{O(d)}$ and NP-hardness proofs. Some problems are left for future work.

1 Introduction

Recently, there has been a substantial amount of research studying computational geometry in the presence of data imprecision [1, 3, 5, 6, 8, 9]. There is quite some variety in the approaches, but they have one thing in common: they focus mostly on planar geometry. The reason is obvious: imprecision makes geometric problems a lot harder to solve, so we start tackling it in the easiest case. However, many real life applications work in 3-dimensional space (and in some cases even higher). Of the work mentioned above, only Bandyopadhyay and Snoeyink [3] focus explicitly on $\mathbb{R}^3$, with protein folding as their application.

Several ways to model imprecision have been proposed. Here we use a general model where each point is represented by a region of possible locations. So, the input is a set of subsets of $\mathbb{R}^d$, and we know that there is one point inside each region.

We study two basic extent measures on points sets: the volume of the axis-aligned bounding box, and the width. These measures give some indication on how "large" the point set is, in a certain sense. In the presence of imprecision, these measures no longer have a fixed value, but a range of possible values. For a given set of imprecise points, our goal is to compute this range. Van Kreveld and Löffler [10] study exactly these questions, in the planar case. Here we investigate whether their solutions still work in higher dimensions, and provide new, more general solutions where they do not.

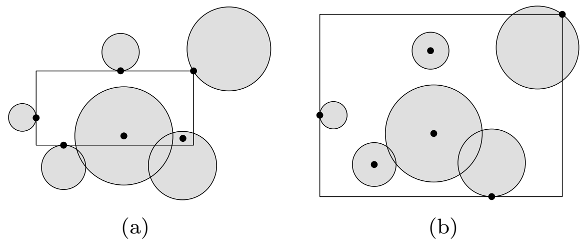


Figure 1: (a) A set of imprecise points as discs in $\mathbb{R}^2$, with points inside them such that the bounding box is as small as possible. (b) The same set of regions, with the points placed such that the bounding box is as large as possible.

1.1 Results

Table 1 shows the results for the two measures we studied, where we model the imprecise points either as a balls or convex polytopes. Because of space restrictions, we only sketch the results here; a detailed discussion of each result can be found in [7]. Additionally, three other measures on point sets are discussed there as well: the diameter, the closest pair, and the smallest enclosing ball.

2 Axis-Aligned Bounding Box

The minimum volume axis-aligned bounding box of a set of points P is the box B of minimum volume with every edge of B parallel to one of the axes in $\mathbb{R}^d$, such that $P \subset B$. We want to find the largest and smallest values that this volume can take. Figure 1 shows an example for the planar case.

2.1 Smallest Axis-Aligned Bounding Box

For an imprecise point set modelled as a set of regions $\mathcal{R}$, the smallest axis-aligned bounding box, is

[*]This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) through the project GOGO.

[†]Department of Information and Computing Sciences, Utrecht University, {hkruger,loffler}@cs.uu.nl

Table 1: Summary of results.

Problem	Model	Smallest	Largest
Axis-aligned Minimum Bounding Box	Convex Polytopes	open	$O(d^2n) + 2^{O(d^2 \log d)}$
	Balls	$O(n^{7+\varepsilon})$ (in $\mathbb{R}^3$)	$O(d^2n) + 2^{O(d^2 \log d)}$
Width	Convex Polytopes	$n^{O(d)}$	NP-hard
	Balls	$O(d^2n^{d+2})$	open

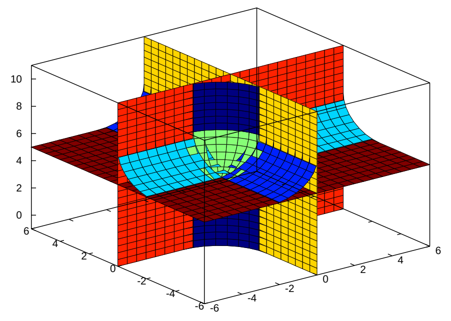


Figure 2: The surface defined by a ball in a corner cell.

the box B of minimum volume, that intersects every $R \in \mathcal{R}$. In each of the d dimensions, consider the minimal width slab that intersects all regions. If any of these slabs has width 0, there is a box of zero volume that intersects all regions, so assume this is not the case. Then the intersections of these slabs forms an axis-aligned box B_0, and we know that at least $B_0 \subset B$. Therefore, we can discard any regions in $\mathcal{R}$ that intersect B_0, and we want to compute the smallest box containing B_0 and intersecting the remaining regions $\mathcal{R}'$. This argument follows Colley *et al.* [4], and was also used in [10].

Now assume $\mathcal{R}$ is a set of balls in $\mathbb{R}^3$. The bounding planes of B_0 partition space into 27 cells. For each ball in $\mathcal{R}'$, we identify the largest subset of $\mathbb{R}^3$ such that if B intersects this set, it also intersects the ball. This set is convex and unbounded, with a monotone surface bounding it on one side. Depending on which cell its centre points lies in, this is a *edge surface* or *corner surface*. (Note that the face cells and the interior of B_0 do not contribute any surfaces.) Figure 2 shows an example of a corner surface. We can prove an important lemma that allows us to assign each region to a unique cell.

Lemma 1 *If the center of a ball $R \subset \mathcal{R}'$ lies in a cell C, then we can restrict its surface to the part inside C.*

If we compute the "lower" envelope of these surfaces in each cell, we obtain 20 single surfaces, each of at most quadratic complexity, and we want to compute the smallest box containing B_0 that intersects all these surfaces.

The smallest box must have some of its corners on these surfaces, since otherwise it could be shrunk. We guess which corners lie on surfaces, and for each we guess on which piece of the surface it lies. There can be at most three such corners, so there are $O(n^6)$ possibilities. For each possibility, we compute the smallest box intersecting them and test in linear time if it intersects all other surfaces. If not, these other surfaces give an extra restriction on the possible boxes. These restrictions could be of non-constant complexity, but we can still compute the optimum in amortised linear time.

Theorem 2 *The total time required to compute the smallest minimum volume axis-aligned bounding box for a set of imprecise points modelled as balls in $\mathbb{R}^3$ is $O(n^{7+\varepsilon})$ for $\varepsilon > 0$.*

This approach does not work when the regions are convex polytopes, because in that case Lemma 1 is no longer true: we can no longer assign each region to a unique cell.

2.2 Largest Axis-Aligned Bounding Box

To compute the largest axis-aligned bounding box for any imprecise point set modelled as a set of regions $\mathcal{R}$ in $\mathbb{R}^d$, it is sufficient to compute the largest axis-aligned bounding box of a subset $\mathcal{R}' \subset \mathcal{R}$ of $4d^2$ regions, specifically the $2d$ most extreme regions in each of the $2d$ axis parallel directions. This is a simple generalisation of [10]. We can prove the following lemma.

Lemma 3 *The largest axis-aligned bounding box of $\mathcal{R}'$ is also the largest axis-aligned bounding box of $\mathcal{R}$.*

Computing $\mathcal{R}'$ takes $O(d^2n)$ time. The largest bounding box of $\mathcal{R}'$ is determined by (at most) $2d$ points, each defining a bounding plane of the box. We test each of the $2^{O(d^2 \log d)}$ possible assignments of regions to bounding planes explicitly.

Theorem 4 *The largest axis-aligned minimum bounding box for any set of imprecise points can be computed in $O(d^2n) + 2^{O(d^2 \log d)}$ time.*

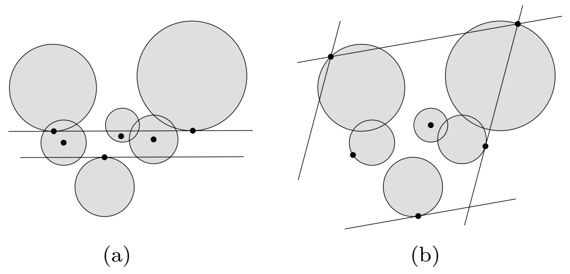


Figure 3: (a) A set of discs in $\mathbb{R}^2$, with points such that the width is as small as possible. (b) The same, but as large as possible.

3 Width

The width of a precise point set $P \subset \mathbb{R}^d$ is defined as the smallest distance between two parallel hyperplanes, such that the entire set P is contained between the two hyperplanes. Figure 3 shows an example of the smallest and largest width of a set of balls.

3.1 Smallest Width

For a set of regions $\mathcal{R}$ in $\mathbb{R}^d$, the minimum width of $\mathcal{R}$ is the same as the smallest slab that intersects all regions. This problem is closely related to the problem of computing a hyperplane transversal of a set of objects in $\mathbb{R}^d$. It is easy to see that the minimum width of $\mathcal{R}$ is zero if and only if there exists a hyperplane that intersects all regions.

Avis and Doskas [2] give an algorithm for computing a hyperplane transversal of a set of polytopes in $\mathbb{R}^d$ in $O(n^d)$ time. For each R_i in a set $\mathcal{R} = \{R_1, \ldots, R_n\}$ of polytopes in $\mathbb{R}^d$, they compute a subdivision Γ_i of the space of normal vectors to supporting hyperplanes of R_i. For each d-dimensional cell σ of Γ_i there exists a unique ordered pair of vertices of R_i such that the hyperplanes defining the smallest width in any direction $v \in \sigma$ pass through these vertices. Clearly, a hyperplane with normal $v \in \sigma$ intersects R_i if and only if it intersects the line segment between these vertices.

To compute a hyperplane transversal of $\mathcal{R}$, they proceed by computing the subdivision $\Gamma^*(\mathcal{R})$ obtained by overlaying all the subdivisions $\Gamma_i, \ldots, \Gamma_n$ and then using linear programming to search for a hyperplane transversal of $\mathcal{R}$ in each cell of $\Gamma^*(\mathcal{R})$. A similar approach can be used to compute the minimum width slab intersecting the regions.

For each cell σ of $\Gamma^*(\mathcal{R})$, we want to compute the minimum width slab with a normal vector $v \in \sigma$. For each region $R_i \in \mathcal{R}$, consider the vertices $p_{\sigma,i}$ and $q_{\sigma,i}$ that are extreme in any direction in σ. Now let P_σ be the convex hull of the set $\{p_{\sigma,i}\}$ and let Q_σ be the convex hull of the set $\{q_{\sigma,i}\}$. We can show it is sufficient to consider all antipodal feature pairs (ψ, ϕ) where ψ is a k-dimensional feature of P_σ and ϕ is a $(d-1-k)$-dimensional feature of Q_σ, such that a pair of parallel hyperplanes with normal vector $v \in \sigma$ are tangent to P_σ and Q_σ at ψ and ϕ respectively. We arrive at the following result for polytopes.

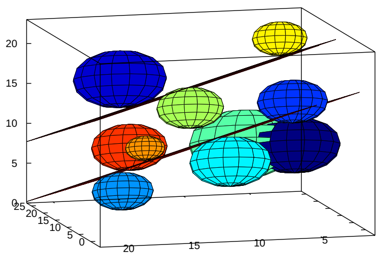


Figure 4: The minimum width of a set of balls in $\mathbb{R}^3$.

Theorem 5 *The minimum width of an imprecise point set modelled as a set $\mathcal{R}$ of polytopes in $\mathbb{R}^d$ can be computed in $O(n^{2d-2})$ time if d is odd or $O(n^{2d-1})$ time if d is even.*

If $\mathcal{R}$ is a set of balls, then it is clearly not possible to construct a subdivision of the space of normal vectors as for polytopes. Thus a completely different approach is needed to compute the minimum width of a set of imprecise points modelled as balls.

We can show that if the minimum width of $\mathcal{R}$ is zero, then there exists a pair of disjoint sets $S_1, S_2 \subseteq \mathcal{R}$ with $1 \leq |S_1|+|S_2| \leq d$ such that a plane h tangent to each ball in $S_1 \cup S_2$ and separating S_1 from S_2 intersects each ball $b \in \mathcal{R}$. Here we focus on the non-zero case, and we can make the following observation.

For a pair of non-empty disjoint subsets $S_1, S_2 \subseteq \mathcal{R}$, with $|S_1| + |S_2| > d$, define H_{S_1,S_2} to be the set of all pairs of parallel hyperplanes (h_1, h_2) tangent to each ball in S_1 and S_2 respectively such that both h_1 and h_2 separate S_1 from S_2.

Lemma 6 *If $\mathcal{R}$ is a set of balls in $\mathbb{R}^d$ with minimum width $W > 0$, then there exists a pair of non-empty disjoint sets $S_1, S_2 \subseteq \mathcal{R}$ with $|S_1| + |S_2| = d + 1$ such that H_{S_1,S_2} is finite and the minimum width of P is given by the distance between a pair of hyperplanes $(h_1, h_2) \in H_{S_1,S_2}$.*

Figure 4 shows a set of balls in $\mathbb{R}^3$ and the pair of parallel planes determining the minimum width. Note that there are two balls tangent to and above the upper plane and two balls tangent to and below the lower plane.

Now, if the minimum width of $\mathcal{R}$ is known to be greater than zero, it can be computed by searching for a pair of sets S_1 and S_2 satisfying Lemma 6. Clearly if S_1 and S_2 satisfy Lemma 6, then $|H_{S_1,S_2}| \leq 2$ (assuming no degeneracies are present in the input). Thus, to compute the minimum width, we can consider all pairs of non-empty disjoint $S_1, S_2 \subseteq \mathcal{R}$ with $|S_1| + |S_2| = d + 1$, determine whether H_{S_1,S_2} is finite, and if it is, test whether $\exists(h_1, h_2) \in H_{S_1,S_2}$ such that each $b \in \mathcal{R}$ has at least one point lying between h_1 and h_2. The minimum distance between all such h_1 and h_2 over all pairs S_1, S_2 will give the minimum width of $\mathcal{R}$.

To compute H_{S_1,S_2}, we need to find all hyperplanes that are tangent to a set of balls. By representing the set of all hyperplanes as a set of points in $\mathbb{R}^d$ using the right duality mapping, the set of hyperplanes tangent to each pair of balls becomes a hypersurface. We then compute the intersection of these surfaces for all pairs of balls from S_1 and S_2.

Testing all sets S_1, S_2 in a mostly brute-force manner, we arrive at the following result.

Theorem 7 *The minimum width of a set of imprecise points modelled as a set $\mathcal{R}$ of balls in $\mathbb{R}^d$ can be computed in $O(d^2n^{d+2})$ time.*

3.2 Largest Width

The maximum width of an imprecise point set modelled as a set of regions $\mathcal{R} = \{R_1, \ldots, R_n\}$ in $\mathbb{R}^d$ is the maximum width over all point sets $P = \{p_1, \ldots, p_n\}$ such that $p_i \in R_i$. Note that this is not the same as the smallest distance between a pair of parallel hyperplanes h_1 and h_2 such that $\mathcal{R}$ is contained between h_1 and h_2.

Computing the maximum width of $\mathcal{R}$ is made difficult by the fact that it is not sufficient to only consider subsets of $\mathcal{R}$: there do exist arbitrarily large sets of imprecise points such that removing any one point reduces the maximum width. Furthermore, it is possible that the maximum width is simultaneously determined by several pairs of parallel hyperplanes [10].

In [10], a proof is given that computing the maximum width of a set of imprecise points modelled as line segments in $\mathbb{R}^2$ is NP-Hard. They showed that for any instance of SAT and some $W > 0$, it is possible to construct a set L of line segments such that the maximum width of L is W if and only if the SAT instance can be satisfied. This construction can be modified to the present case for polytopes.

In $\mathbb{R}^2$ we can use the same construction and replace each line segment with a triangle of height ε (for some very small ε). The resulting set of convex polygons has maximum width W if and only if the SAT instance can be satisfied. We can extend this to a set of polytopes in $\mathbb{R}^d$ by using the first two dimensions for the planar construction, and extending the polytopes by at least $2W$ in all other dimensions.

Theorem 8 *For any instance of SAT and any $W > 0$, it is possible to construct a set $\mathcal{R}$ of convex polytopes in $\mathbb{R}^d$ such that the maximum width of $\mathcal{R}$ is W if and only if the SAT instance can be satisfied.*

For the case where the regions are balls, such a construction does not work; thus, this problem remains open.

4 Conclusions and Open Problems

We studied two extent measures on point sets in higher dimensions in the presence of data imprecision. We obtained some first results, but there is still a lot of room for improvement: better running times may be possible in some cases, and some variants remain open. Apart from this, there has been very little work on high dimensional geometric problems in an imprecise setting, and there are plenty of possibilities for future work.

References

[1] M. Abellanas, F. Hurtado, and P. Ramos. Structural tolerance and Delaunay triangulation. *Information Processing Letters*, 71:221–227, 1999.

[2] D. Avis and M. Doskas. Algorithms for high dimensional stabbing problems. *Discrete Applied Mathematics*, 27:39–48, 1990.

[3] D. Bandyopadhyay and J. Snoeyink. Almost-Delaunay Simplices: Nearest Neighbor Relations for Imprecise Points. In *PROC. 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 410–419, 2004.

[4] P. Colley, H. Meijer, and D. Rappaport. Optimal nearly-similar polygon stabbers of convex polygons. In *Proceedings of the 6th Canadian Conference on Computational Geometry*, pages 269–274, 1994.

[5] L. Guibas, D. Salesin, and S. J. Constructing Strongly Convex Approximate Hulls with Inaccurate Primitives. *Algorithmica*, 9:534–560, 1993.

[6] A. A. Khanban and A. Edalat. Computing Delaunay Triangulation with Imprecise Input Data. In *Proc. 15th Canadian Conference on Computational Geometry*, pages 94–97, 2003.

[7] H. Kruger. Basic measures for imprecise point sets in $\mathbb{R}^d$. Master's thesis, Utrecht University, 2008.

[8] M. Löffler and J. Snoeyink. Delaunay Triangulations of Imprecise Points in Linear Time after Preprocessing. In *Annual Symposium on Computational Geometry*, pages 298–304. ACM, 2008.

[9] T. Nagai and N. Tokura. Tight Error Bounds of Geometric Problems on Convex Objects with Imprecise Coordinates. In *Japanese Conference on Discrete and Computational Geometry*, pages 252–263, 2000.

[10] M. van Kreveld and M. Löffler. Largest bounding box, smallest diameter, and related problems on imprecise points. In *Proc. 10th Workshop on Algorithms and Data Structures*, LNCS 4619, pages 447–458, 2007.

Some Regularity Measures for Convex Polygons

Ferran Hurtado[*] Vera Sacristán[*] Maria Saumell[*]

Abstract

We propose new measures to evaluate to which extent the shape of a given convex polygon is close to the shape of some regular polygon. We prove that our parameters satisfy several reasonable requirements and provide algorithms for their efficient computation. The properties we mostly focus on are the facts that regular polygons are equilateral, equiangular and have radial symmetry.

1 Introduction

Regular polygons have fascinated mathematicians of all times. As an example, let us mention that, despite the numerous mathematical contributions Gauss made throughout his life, he was so proud of his compass and straight-edge construction of the regular heptadecagon that he asked for one to be inscribed on his tombstone.

In addition to the mathematical interest, a considerable amount of manufactured objects are regular-shaped, for which reason some methods to detect regular polygons in given images have been developed in recent years (see [2] or [5]). Our approach is related to the one in these works because we want to propose parameters that give us some information about how regular a convex polygon is. An antecedent of this problem that has received substantial attention lately is the study of convexity coefficients for polygons (see, for example, [3, 11]).

Although one could come up with many criteria to measure whether a convex polygon is close to being regular, we have focused on measures which, besides fulfilling some natural conditions, correspond to a computational geometry perspective. Following the rational from [11], we require several conditions on the parameters: i) the regularity measure is a number from $(0,1]$; ii) the regularity measure of a given polygon equals 1 if and only if it is regular; iii) there are polygons whose regularity measure is arbitrary close to 0; iv) the regularity measure of a polygon is invariant under similarity transformations.

While our research is theoretically oriented, let us mention that it is to some extent related to applications in metrology, robotics and discretizations of planar domains into suitable meshes. In the three cases, the measurements we present can be seen as quality control tests to check the regularity of an object whenever that is the desired shape.

Additional problems, details, and full proofs, can be found in [9].

2 Polygons with the same number of edges

A natural constraint for evaluating whether a given convex n-gon P (we also assume that P has no straight angles) is close to being regular is to compare it to a regular polygon with the same number of edges. A simple way could be to compare P to a regular n-gon with the same area or the same perimeter. More sophisticated methods can nevertheless be put forward as we will see.

The main parameter proposed in this section is computed in two steps, through which P is transformed into a regular polygon R. The regularity measure captures the differences between P and R.

2.1 Inscription in a circle

In the first step, we transform P into a polygon inscribed in a circle and we scale the two objects so that the radius of the circle is one. In order to do that, we give a method to compute a positive real number $r(P)$ and a polygon P' such that P' has the same ordered sequence of edge lengths as P and all its vertices lie on a circle $C(P)$ of radius $r(P)$. Let us denote the ordered sequence of edge lengths of P by $l_1, l_2, \ldots, l_n$ and assume that l_1 corresponds to the longest edge. Theorem 1 shows that there always exists such a polygon P' and that $r(P)$ is the solution of an equation depending only on P.

Theorem 1 [6] *For any given convex n-gon P, the following three statements are equivalent:*

(i) *the center of $C(P)$ is contained in P';*

(ii) $\sum_{i=1}^{n} 2\arcsin(l_i/l_1) \geq 2\pi$;

(iii) *$r(P)$ is the unique solution of* $\sum_{i=1}^{n} 2\arcsin(l_i/2r) = 2\pi$.

These three statements are also equivalent:

(i) *the center of $C(P)$ is exterior to P';*

[*]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Barcelona, Spain, {Ferran.Hurtado,Vera.Sacristan,Maria.Saumell}@upc.edu. Partially supported by projects MEC MTM2006-01267 and DURSI GenCat 2005SGR00692.

(ii) $\sum_{i=1}^{n} 2\arcsin(l_i/l_1) < 2\pi$;

(iii) $r(P)$ *is the unique solution of* $\sum_{i=2}^{n} \arcsin(l_i/2r) = \arcsin(l_1/2r)$.

An approximated value of $r(P)$ can be obtained by numerically solving the corresponding equation. The following bounds might also be taken into account:

Proposition 2 *Given P, it holds that:*

(i) *The radius $r(P)$ is so that*

$$r(P) \geq \max\left\{\frac{l_1}{2}, \frac{\sum_{i=1}^{n} l_i/2n}{\sin(\pi/n)}\right\}$$

and there exist examples where these values are achieved.

(ii) *If the inscription of P' in $C(P)$ is such that the center of $C(P)$ is contained in P', then*

$$r(P) \leq \frac{\sum_{i=1}^{n} l_i}{4}$$

and we can construct a family of inscribed polygons such that the radius of the circumscribed circles get arbitrarily close to this bound.

The first lower bound is trivial and the second one results from the fact that, among all n-gons inscribed in a given circle, the maximum perimeter is achieved by the regular one. The upper bound is a consequence of the relation between the lengths of a chord of a circle and its corresponding arc.

Now let $\alpha(P, P')$ be the absolute value of the maximum difference between one interior angle of P and its corresponding interior angle in P'. We define the parameter $\mu_1^1(P) = (\pi - \alpha(P, P'))/\pi$, which will be used later.

2.2 Search of regular position

Next we present the second step of the transformation of P. We will make the vertices $p_1, p_2, \ldots, p_n$ of P' slide along the circle until they get to the position of the vertices $r_1, r_2, \ldots, r_n$ of a regular polygon R (see Fig. 1a for an example), minimizing the distance covered by the vertex that travels the most. We will assume that the vertices of all polygons are always given in counterclockwise order. We say that p_i and r_j are *matched* if p_i travels to r_j to reach the regular position. The next lemma offers a nice characterization of one of the solutions:

Lemma 3 *There exists a solution in which p_i and r_i are matched for all i.*

The key idea of the proof is that two vertices of P' cannot meet during their movement towards the regular position if the movement is to be minimized.

Let us now suppose that $r_1, r_2, \ldots, r_n$ slide counterclockwise along the circle with constant speed of 1 rad/s from the position in which p_1 and r_1 are coincident until one complete turn has been done. If $d_1(t)$ denotes the function that gives the distance on the circle between p_1 and r_1 at time t, we have $d_1(t) = t$, for $t \in [0, \pi]$, and $d_1(t) = 2\pi - t$, for $t \in (\pi, 2\pi)$.

Moreover, all the analogously defined functions $d_i(t)$, $i \neq 1$, are translations of $d_1(t)$ and can be computed in $\theta(n)$ time. The optimal position that we search is then given by the point of minimum y-coordinate in the upper envelope of the arrangement of the functions $d_i(t)$. See Fig. 1b for an example. We then have:

Theorem 4 *Given P', the problem of computing the movement from P' into a regular position that minimizes the maximum distance $d(P', R)$ covered by a vertex can be solved in $O(n \log n)$ time.*

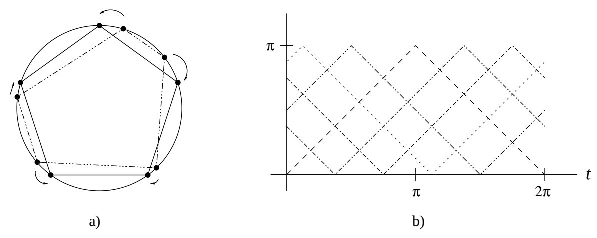


Figure 1: a) A movement from P', into a regular position. b) Graphical representation of the functions $d_i(t)$.

Notice that $d(P', R) < \pi - \pi/n$. We define $\mu_1^2(P) = (\pi - \pi/n - d(P', R))/(\pi - \pi/n)$ and $\mu_1(P) = \mu_1^1(P)\,\mu_1^2(P)$ (other measure combinations are possible). Then $\mu_1(P)$ fulfills conditions i)-iv).

Observation 1 *In this context, an alternative parameter to be minimized is the sum of the distances covered by all the vertices of P'. It can also be computed in $O(n \log n)$ time and lead to another regularity measure satisfying the requirements i)-iv).*

3 Inscription and circumscription

There exist convex polygons which are very similar to regular polygons with a different number of edges. In this section we propose suitable regularity measures to cope with these situations.

Let R stand for a regular m-gon. We compute the smallest similar copy R_E of R enclosing P (*inscription problem*) and the largest similar copy R_I of R enclosed in P (*circumscription problem*), allowing translations, rotations and scaling (see the examples in Fig. 2). In this case, the regularity coefficients (perhaps m-regularity coefficients would be a more appropriate expression) are $\mu_2(P) = \text{area}(P)/\text{area}(R_E)$

and $\mu_3(P) = \text{area}(R_I)/\text{area}(P)$, and they both satisfy i)-iv).

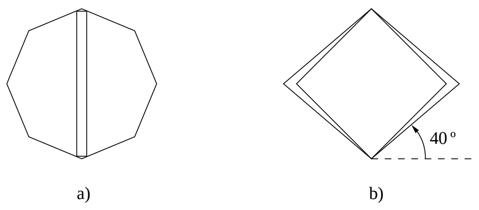


Figure 2: a) Inscription of a rectangle in a regular octagon. b) Circumscription of a rhombus to a square.

We will sketch how to solve these containment problems. For the sake of brevity, only the inscription problem will be discussed.

It is obvious that there must be some incidences between P and R_E, and that these incidences can be of different types. Our first attempt to solve the inscription problem has been to separately obtain the three types of solution candidates: those having an edge in contact with an edge of P, those having a vertex in contact with a vertex of P and those not having any of the previous special contacts. However, we have obtained better results with another technique that is related to a more general version of the problem.

If the exterior polygon Q is only required to be convex, it has been shown in [1] that the smallest enclosing similar copy of Q can be computed in $O(mn^2 \log n)$ time. The main idea behind it is that, if we fix Q and look for the largest copy of P inside it, the space of all similar placements of P enclosed in Q is a convex polytope in $\mathbb{R}^4$ given by the intersection of mn halfspaces $L_{i,j}$. Then, the maximum size is achieved by a vertex of the polytope (see [1] for details).

We have been able to prove that the complexity of the polytope is much lower in our case when $Q = R_E$ because it suffices to consider rotations of angles in $[0, 2\pi/m]$. The argument is as follows: each halfspace $L_{i,j}$ forces the vertex i of P to lie in the same halfplane defined by the prolongation of the edge j of R_E as R_E does. Because of the regularity of R_E, P only needs to rotate $2\pi/m$ and the number of restrictions $L_{i,j}$ that must be taken into account can be proven to reduce to $n+m$. Since the intersection of $O(n+m)$ halfspaces in $\mathbb{R}^4$ has $O((n+m)^2)$ vertices that can be computed in $O((n+m)^2)$ time (see [8]), we obtain the following result:

Theorem 5 *The inscription problem can be solved in $O((n+m)^2)$ time.*

4 Radial symmetry

Radial symmetry is yet another main characteristic of regular polygons that, in fact, has attracted attention even very recently [7]. In this section we deal with properties of regular polygons that are a consequence of their symmetry and propose measures of the regularity of a polygon based on them.

4.1 Equiangular distribution

Our aim is to solve the following problem: given the convex n-gon P, the pencil of n half-lines $\{t_i\}_i$ with the origin at the barycenter of P and slopes $\{\tan 2\pi i/n\}_i$ splits the area of P into n pieces (see Fig. 3a). We seek to find the rotation angle φ_0 around the barycenter that minimizes the area of the smallest portion.

Let us denote the barycenter of P by G; we can assume that $G = (0,0)$. We define $a_i(\varphi)$, $\varphi \in [0, 2\pi)$, as the area of the portion of P delimited by t_i and t_{i+1} when t_1 and the x-axis form angle φ. The following lemma describes the functions $a_i(\varphi)$; it can be proved by splitting each portion of P into suitable triangles.

Lemma 6 *The functions $a_i(\varphi)$ are the continuous concatenation of $O(n)$ pieces of branches of rational functions that are quotients of two second degree polynomials whose variable is $\tan\varphi$. Each can be computed in $O(n)$ time.*

The symmetry of the pencil allows us to restrict our search to $\varphi \in [0, 2\pi/n]$. We are interested in the arrangement of the functions $a_i(\varphi)$ in this interval, which can be seen as a set of $O(n)$ functions defined on possibly smaller intervals, each pair of which intersects in at most 4 points. Consequently, the complexity of the lower envelope is at most $\lambda_6(n) = O(n \cdot 2^{O(\alpha^2(n))})$ [10], where $\lambda_s(k)$ denotes the maximal complexity of the upper or lower envelope of a set of k curves which pairwise intersect at most s times and $\alpha(n)$ is the functional inverse of Ackermann's function. We have proved:

Theorem 7 *The angle φ_0 can be computed in $O(\lambda_6(n) \log n)$ time.*

If $a_j(\varphi_0)$ is this smallest portion, $\mu_4(P) = n\, a_j(\varphi_0)/\text{area}(P)$ is a regularity measure satisfying conditions i), iii) and iv). The next result, which can be checked by studying $da_1(\varphi)/d\varphi$ ensures that it also fulfills ii).

Proposition 8 *The function $a_1(\varphi)$ is constant if, and only if, P is regular.*

Observation 2 *Other angles that give rise to related coefficients are the angle that maximizes the area of the largest portion and the angle that maximizes the difference between the areas of the largest and the smallest part. Notice that they can similarly be obtained from the arrangement of the functions $a_i(\varphi)$.*

4.2 Area center

In this subsection we continue investigating whether the area of P can easily be divided into similar size pieces (see Fig. 3b). More precisely, we want to find a point q in P minimizing the parameter $\max_i\{\,\text{area}\,(\triangle\, qp_ip_{i+1})\}$.

This problem can be solved by means of Voronoi diagrams. Given a point (x, y) and a segment e, we define the distance $\tilde{d}((x, y), e)$ as the area of the triangle which base is e and which third vertex is (x, y). If we define the further Voronoi diagram (FVD(P)) of the edges of P for the distance $\tilde{d}$, then is not difficult to prove that

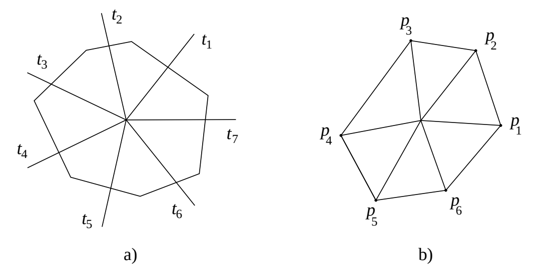


Figure 3: Divisions of convex polygons into pieces.

Theorem 9 *The center q is either a vertex of FVD(P) or the intersection point of an edge of FVD(P) and the boundary of P.*

Therefore we can solve our problem if we are able to compute FVD(P). This construction can be done by a divide-and-conquer algorithm in which the main difficulty lies in proving the merge step to be linear. Then it holds:

Theorem 10 *Given P, its FVD(P) can be computed in $O(n \log n)$ time.*

This problem can be solved more efficiently by immersion in $\mathbb{R}^3$. Place the polygon P in $z = 0$ and, for each edge e_i of P, consider a plane Π_i through e_i and such that the distance $\tilde{d}$ between a point p of P and e_i equals the vertical distance between p and Π_i. The center q is the orthogonal projection onto $z = 0$ of the point of minimum z-coordinate in the intersection of the upper half-spaces defined by the planes Π_i. As this is a linear programming problem, we have (see [4]):

Theorem 11 *The problem of finding the center q can be solved in $O(n)$ time.*

Now let $d(\cdot,\cdot)$ denote the Euclidean distance in $\mathbb{R}^2$ and let i_1, i_2 and i_3 be such that $T_{i_1} = \text{area}\,(\triangle\, qp_{i_1}p_{i_1+1}) = \max_i\{\,\text{area}\,(\triangle\, qp_ip_{i+1})\}$, $d_{i_2} = d(q, p_{i_2}) = \max_i\{d(q, p_i)\}$ and $d_{i_3} = d(q, p_{i_3}) = \min_i\{d(q, p_i)\}$. Then $\mu_5(P) = d_{i_3}/d_{i_2} \cdot \text{area}\,(P)/(n\, T_{i_1})$ is a regularity measure that satisfies i)-iv).

Observation 3 *The point that maximizes the area of the smallest triangle and the point that minimizes the difference between the areas of the largest triangle and the smallest one are useful alternatives for this type of measures. Optimal algorithms to compute them are given in [9].*

5 Conclusions and further work

We have presented several measures of regularity for convex polygons that satisfy the requirements described in the introduction, that are efficiently computable, and that cover a wide range of approaches to evaluate how far a convex polygon is from being regular.

While we have focused on a theoretical perspective, it would be interesting in the future to have some experimental results at our disposal to check which (combinations of) measures give regularity descriptions that are the closer to our intuition about the topic.

References

[1] P. K. Agarwal, N. Amenta and M. Sharir. Largest Placement of One Convex Polygon inside Another. *Discrete Comput. Geom.*, 19:95–104, 1998.

[2] N. Barnes, G. Loy, D. Shaw and A. Robles-Kelly. Regular Polygon Detection. *Proc. ICCV'05*, China, 2005.

[3] L. Boxer. Computing Deviations from Convexity in Polygons. *Pattern Recog. Lett.*, 14:163–167, 1993.

[4] N. Megiddo. Linear Programming in Linear Time When the Dimension Is Fixed. *J. ACM*, 31:114–127, 1984.

[5] G. Piccioli, E. D. Micheli, P. Parodi and M. Campani. Robust Method for Road Sign Detection and Recognition. *Image Vision Comput.*, 14:209–223, 1996.

[6] I. Pinelis. Cyclic Polygons with Given Edge Lengths: Existence and Uniqueness. *J. Geom.*, 82:156–171, 2005.

[7] I. Pinelis. A Characterization of the Convexity of Cyclic Polygons in Terms of the Central Angles. *J. Geom.*, 87:106–119, 2007.

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

[9] M. Saumell. *Mesures de regularitat per a polígons convexos.* Master Thesis. Universitat Politècnica de Catalunya, 2008.

[10] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, 1995.

[11] J. Žunić and P. L. Rosin. A Convexity Measurement for Polygons. *Proc. BMVC'02*, United Kingdom, 2002.

Inducing n-gon of an arrangement of lines

Ludmila Scharf* Marc Scherfenberg*

Abstract

We show that an arrangement $\mathcal{A}$ of n lines in general position in the plane with $n \geq 3$ has an inducing polygon of size n. The proof is constructive, that is we describe an algorithm that constructs a simple n-gon inducing $\mathcal{A}$.

1 Introduction

Every simple polygon induces an arrangement of lines, simply by extending its edges. We consider the question whether every arrangement of lines in the plane has an *inducing polygon*, namely, a simple polygon P such that every line l of the arrangement $\mathcal{A}$ is collinear with *one* edge of P and that every edge of P is collinear with some line of $\mathcal{A}$, see Fig. 1 for an example. There are arrangements that cannot be induced by a simple polygon. Lines that all intersect in one point and lines that form a 3×2 parallel grid serve as examples of such arrangements. However, we will show that when the lines of an arrangement are *in general position*, i.e., no three lines intersect in one point, and no two lines are parallel, an inducing polygon with n edges exists and can be found in $O(n^2)$ time. From now on we consider only arrangements of lines in general position in the plane.

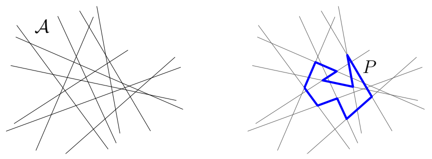


Figure 1: Line arrangement $\mathcal{A}$ and a simple polygon P inducing $\mathcal{A}$.

This problem has been addressed by Bose et al. in [1]. It is known that a simple inducing polygonal path with n edges always exists, and for some classes of arrangements an inducing n-gon exists, see [1, 4, 6]. Also, for a weaker variant of the problem, namely, if the polygon is allowed to have collinear edges, i.e., more than one edge on one line, we proved in [6] that an inducing polygon of size $O(n)$ always exists.

In this paper we describe an incremental algorithm that constructs for an arrangement $\mathcal{A}$ of n lines in general position in the plane a simple n-gon inducing $\mathcal{A}$.

*Institute of Computer Science, Freie Universität Berlin, {scharf,scherfen}@mi.fu-berlin.de

2 The Algorithm

Let $\mathcal{A}$ be an arrangement of n lines in the plane, where $n \geq 3$. Our algorithm constructs an inducing polygon P of size n for the arrangement $\mathcal{A}$ incrementally.

2.1 Initialization

For the initialization take two arbitrary lines g_1, g_2 such that their intersection point p is the first intersection point on both lines, e.g., choose p as the intersection point of the arrangement with the maximum x-coordinate. Define the orientation of these two lines in direction from p towards the remaining intersection points. Without loss of generality we can assume that all intersection points of g_2 lie in the positive half-plane of g_1, denoted by $H^+(g_1)$, see Fig. 2.

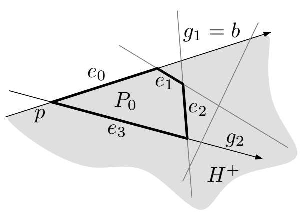


Figure 2: Initialisation and edge labeling.

Let P_0 be the convex polygon consisting of the boundary of the bounded cell in $\mathcal{A}$ which is incident to g_1, g_2, and p. Let P_i denote the polygon constructed in step i, $\mathcal{I}(\cdot)$ the lines induced by a polygon or a polygonal chain, and $\mathcal{A}_i = \mathcal{A} \setminus \mathcal{I}(P_i)$ the arrangement without the lines induced by P_i. Note that P_0 is completely contained in an unbounded cell of $\mathcal{A}_0$. It is an invariant that is maintained in every extension step of the algorithm. Let $R^{(i)}$ denote the boundary of the unbounded cell $C^{(i)}$ in $\mathcal{A}_i$ which contains P_i. We also maintain a so called *base line* $b^{(i)}$. For P_0 the base line is $b^{(0)} = g_1$.

2.2 Incremental Construction

We first introduce some notation, which is used in the description of incremental steps. The edges of P_i are enumerated in the following way: the edge contained in the base line $b^{(i)}$ is the edge $e_0^{(i)}$. In counter-clockwise order we enumerate with negative indices the edges contained in the previous base lines $e_{-1}^{(i)}, \ldots, e_{-m}^{(i)}$, where $e_{-m}^{(i)}$ is contained in the first base line $b^{(0)}$. Note that $m \leq i$, since the base line is not necessarily changed in every step. The remaining edges are enumerated in clockwise order with positive indices $e_1^{(i)}, \ldots, e_k^{(i)}$, where $e_1^{(i)}$ is incident to $e_0^{(i)}$ and $e_k^{(i)}$ is incident to $e_{-m}^{(i)}$. We call the edges contained in base lines *base edges* and other edges *non-base edges.*

A line containing an edge $e_j^{(i)}$ is denoted by $l_j^{(i)}$, and the intersection point of two lines $l_j^{(i)}, l_m^{(i)}$ by $x_{j,m}^{(i)}$. We define the orientation of base edges in clockwise direction, and the orientation of non-base edges in counter clockwise direction with respect to the polygon P_i. For each line $l_j^{(i)}$ its orientation is defined by the orientation of the edge $e_j^{(i)}$. The part of $l_j^{(i)} \setminus e_j^{(i)}$ oriented in positive (negative) direction of $l_j^{(i)}$ is called positive (negative) half-line and is denoted by $l_j^{(i)+}$ ($l_j^{(i)-}$). The intersection point of the positive or negative half-line of a line $l_j^{(i)}$ with the boundary $R^{(i)}$ is denoted by $x_{R,j}^{(i)+}$ or $x_{R,j}^{(i)-}$, respectively, and the part of $R^{(i)}$ between the intersection points with the lines $l_j^{(i)}, l_k^{(i)}$ is denoted by $R_{j,k}^{(i)}$.

Note that for the base line $b^{(i)}$ we maintain the property that all intersection points of $b^{(i)}$ with $\mathcal{A}_i$ lie in $b^{(i)+}$. Thus, the intersection point $b^{(i)} \cap R^{(i)}$ is always the point $x_{R,0}^{(i)+}$, which will be denoted by $x_{R,0}^{(i)}$. Finally, for simplicity we will omit the index (i) if all identifiers refer to the same step i, and will use the index in order to distinguish between the notation in different steps.

The algorithm performs one of the extension constructions described below until after some i-th extension step all lines of $\mathcal{A}$ are induced by P_i. The inducing polygon for $\mathcal{A}$ is then $P = P_i$.

We consider three non-base edges e_1, e_2, e_3, the concave chain of base edges, and the convex chain $R^{(i)}$ – the boundary of the cell containing P_i. All possible combinatorial configurations of this finite set of objects are partitioned into seven cases. For each of the cases we specify the incremental construction of P_{i+1} from P_i. Observe that after every extension step the constructed polygon P_i lies completely on one side of the line l_1, i.e., the edge e_1 is an edge of the convex hull of P_i.

The first case distinction is whether the negative half-line of l_1 intersects the boundary R. If it does, Case 1 applies.

Case 1: [$l_1^{(i)-} \cap R^{(i)} \neq \emptyset$] The edge e_1 is removed from P_i and is substituted by the chain $x_{0,1}$, $x_{R,0}$, $R_{0,1}$, $x_{R,1}^-$, $x_{1,2}$. The base line for P_{i+1} remains unchanged $b^{(i+1)} = b^{(i)}$, and the edge $e_1^{(i+1)}$ is the edge of $R_{0,1}^{(i)}$ incident to $b^{(i)}$, see Fig. 3.

If l_1^- does not intersect R, then l_1^+ has to. Therefore, for the remaining cases the condition (C1) is fulfilled:

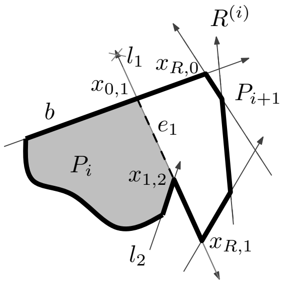


Figure 3: Extension Case 1: The polygon P_i is the shaded area. The identifiers refer to P_i, and the new polygon P_{i+1} is outlined by the bold black line.

$$l_1^{(i)-} \cap R^{(i)} = \emptyset \ . \tag{C1}$$

The next distinction is whether l_2^+ intersects R:

Case 2: [(C1) and $l_2^+ \cap R \neq \emptyset$] The edge e_1 is removed from P_i and is substituted by the chain $x_{0,1}$, $x_{R,1}^+$, $R_{1,2}$, $x_{R,2}^+$, $x_{1,2}$, see Fig. 4. The base line for the polygon P_{i+1} is now $b^{(i+1)} = l_1^{(i)}$, and the edge $e_1^{(i+1)}$ is the edge of $R_{12}^{(i)}$ incident to $l_1^{(i)}$.

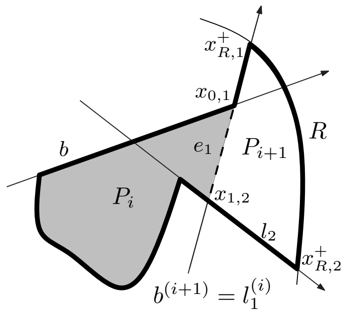


Figure 4: Extension Case 2.

For the remaining cases condition (C2) holds:

$$l_2^{(i)+} \cap R^{(i)} = \emptyset \ . \tag{C2}$$

It can be shown that then the intersection of $l_2^{(i)-}$ with the polygon P_i is exactly one point $x_{j,2}, j \leq 0$, on a base edge. One of the next two cases applies if

$$l_2^{(i)-} \cap P_i \in e_0^{(i)} \ . \tag{C3}$$

The difference between the two cases is whether the point $x_{1,3}$ lies inside or outside the cell C.

Case 3: [(C1) and (C2) and (C3) and $x_{1,3} \in C_R$] The edges e_1, e_2, e_3 and the part $x_{0,2}x_{0,1}$ of the edge e_0 are removed from P_i. The new added chain is $x_{0,2}$, $R_{2,1}$, $x_{1,3}$, $x_{3,4}$, see Fig. 5. The base line for the polygon P_{i+1} is now $b^{(i+1)} = l_2^{(i)}$.

Case 4: [(C1) and (C2) and (C3) and $x_{1,3} \notin C_R$] If $\mathcal{A}_i$ is induced by $R_{1,3}$, P_i induces the complete arrangement up to the lines in $R_{1,3}$, we call it a *finishing case* (see below). For now assume that

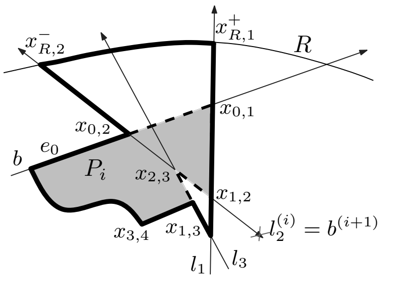


Figure 5: Extension Case 3.

$\mathcal{A}'_i = \mathcal{A}_i \setminus \mathcal{I}(R_{1,3})$ is not empty. Let the boundary of the unbounded face of $\mathcal{A}'_i$ which contains P_i be denoted by R', the intersection points of l_1^+ and l_2^- with R' by $x'_{R,1}$ and $x'_{R,2}$, respectively, and the part of R' between $x'_{R,2}, x'_{R,1}$ by $R'_{2,1}$.

For the extension of the polygon we again remove the part $x_{0,2}x_{0,1}x_{1,2}x_{2,3}$ and substitute it by $x_{0,2}, R'_{2,1}, R_{1,3}, x_{2,3}$, see Fig. 6. The base line for the polygon P_{i+1} is now $b^{(i+1)} = l_2^{(i)}$.

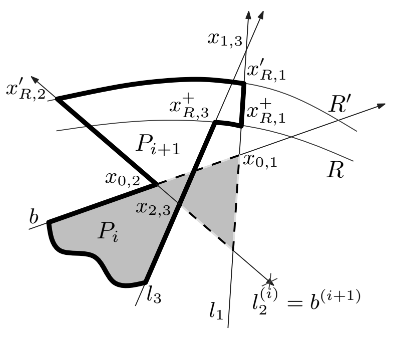


Figure 6: Extension Case 4.

The next case applies if the intersection point $l_2^- \cap P_i$ lies on a base edge e_j with $j \leq -2$.

Case 5: [(C1) and (C2) and $l_2^- \cap P_i \in e_j, j \leq -2$] We remove the edge e_{j+1} by extending its adjacent edges e_j and e_{j+2} up to the point $x_{j,j+2}$. Additionally, the segment $x_{j+1,1}x_{0,1}$ of e_1 is replaced by $x_{j+1,1}$, $R_{j+1,0}$, $x_{0,1}$, see Fig. 7. The base line for the polygon P_{i+1} remains unchanged, i.e. $b^{(i+1)} = b^{(i)}$. Note that the new edge on the line $l_{j+1}^{(i)}$ is not considered to be a base edge in all subsequent steps.

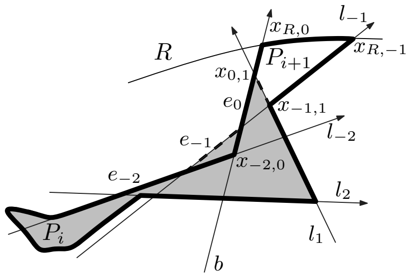


Figure 7: Extension Case 5.

Finally, if the condition

$$l_2^- \cap P_i \in e_{-1} \tag{C4}$$

is fulfilled, we distinguish whether at least one of the points $x_{0,3}, x_{1,3}$ lies inside the cell C or not. We can show that in the last two cases the point $x_{2,3}$ lies in the segment $x_{-1,2}x_{0,2}$.

Case 6: [(C1), (C2), (C4), $\{x_{0,3}, x_{1,3}\} \cap C_R \neq \emptyset$] If both $x_{0,3}$ and $x_{1,3}$ lie inside the cell C, let y denote the one that is closer to e_3, otherwise, $y = \{x_{0,3}, x_{1,3}\} \cap C$. Let g be the line intersecting l_3 at y, and h the other one of the two lines l_0, l_1. We cut the polygon P_i at the points $x_{-1,2}$ and $x_{3,4}$ and discard the chunk containing e_0. Instead, the edge e_3 is replaced by the segment $x_{3,4}y$. Further, we add $yx_{0,1}$ of line g, $x_{0,1}x_{R,h}$ of line h, $R_{h,2}$ and the segment $x_{R,2}x_{-1,2}$, see Fig. 8. The base line for the polygon P_{i+1} is now $b^{(i+1)} = l_2^{(i)}$.

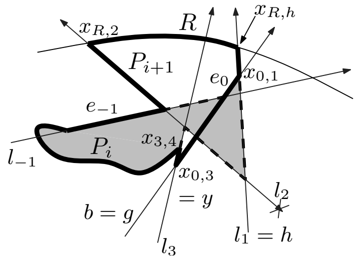


Figure 8: Extension Case 6 illustrated for $g = l_0$.

Case 7: [(C1), (C2), (C4) and $\{x_{0,3}, x_{1,3}\} \cap C_R = \emptyset$] If $\mathcal{A}'_i = \mathcal{A}_i \setminus \mathcal{I}(R_{13})$ is empty we have a finishing case. Otherwise, we proceed similar to Case 4. Let R' denote the boundary of the unbounded face in $\mathcal{A}'$ containing P_i, and let $R'_{2,0}$ denote the part of R' between the intersection points $x'_{R,2}, x'_{R,0}$ with the half-lines b^+ and l_2^- respectively. The polygon P_i is cut at the points $x_{-1,2}$ and $x_{2,3}$ and the chunk containing e_0 is discarded. Instead, the polygon is completed by the following chain: $x_{-1,2}$, $R'_{2,0}$, $x_{0,1}$, $R_{1,3}$, $x_{2,3}$, see Fig. 9. The base line of the polygon P_{i+1} is $b^{(i+1)} = l_2^{(i)}$.

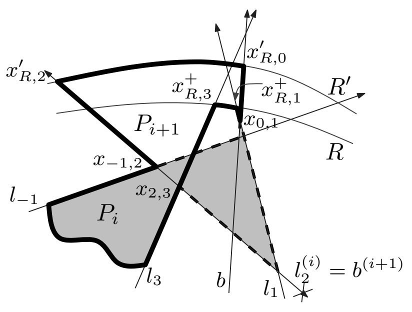


Figure 9: Extension Case 7.

Finishing cases: The finishing configurations for Case 4 and 7 extensions are almost identical and involve six further case distinctions where we additionally consider the position of lines l_4 and l_5. Full description of the extension cases and the complete correctness proof can be found in [5].

2.3 Correctness

For the correctness of the algorithm we need to show that the constructed polygon P is simple, every line in $\mathcal{A}$ is collinear to exactly one edge in P and every edge in P is collinear to some line in $\mathcal{A}$.

The extension steps are repeated as long as lines exist that are not induced by P. After every extension step P_{i+1} induces at least one line that is not induced by P_i. Every line induced by P_i is also induced by P_{i+1}. Thus, the algorithm terminates after at most n extensions, and every line of $\mathcal{A}$ is induced by the constructed polygon. Furthermore, in every construction step the modification chain contains exactly one segment on each affected line.

For the simplicity of the polygon we show that, assuming P_i is simple, for every extension case the constructed polygon P_{i+1} is also simple. The initial polygon P_0 is clearly simple. The new added chain in every extension step is simple by construction. Then if the new edges do not intersect the unmodified edges, the resulting polygon is simple. In the following we only sketch the idea without proving the claims made.

Considering the general structure of the constructed polygon we observe that the base edges form a concave chain and the non-base edges are contained in the union of the positive half-plains of the base lines.

The line l_1 has no intersection with the polygon except for the edge e_1 and there are no edges of the polygon in the positive half-plane of l_1. Thus, *Case 1 and 2 extensions are simple.*

If the Conditions (C1) and (C2) are fulfilled, then the only intersections of the line l_2 with the polygon are the edge e_2 and exactly one point on the chain of base edges. Thus, the described cases cover all possible configurations. Additionally, the negative half-plane of l_2, $H^-(l_2)$, contains only the edge e_1 and the part of the chain of base edges between the point $x_{0,1}$ and the intersection point with l_2. In Case 4, 5, and 7 these are exactly the edges that are removed. The new added chain lies completely in $H^-(l_2)$, therefore, *Case 4, 5, and 7 extensions are simple.*

Furthermore, if the Conditions (C1) and (C2) are fulfilled, then the last executed extension step was Case 1 with exactly one new line added. If additionally $l_2 \cap P_i \in e_{-1}$, then the extension preceding Case 1 was a Case 2 extension with exactly one new line. Finally, if the conditions of Case 5 apply, the last extension steps were a sequence of Case 2 followed by a Case 1 extension, where in each step exactly one new line was added.

For every non-base line l_m it holds that either l_m was a base line at some step and was changed to a non-base line in a Case 5 extension, or the intersection of l_m with every non-base edge e_k with $k > m$ is empty. Combining this fact with the above case backtracking we can show that the line l_3 in Case 3 and 6 cannot be a former base line. Then in Case 3 and 6 the positive half-plane of l_3 does not contain any unmodified edge. The new added chain lies completely in $H^-(l_2) \cup H^+(l_3)$. Thus, *Case 3 and 6 extensions are simple.*

Applying the above fact to line l_4 in the finishing cases we can show that the finishing cases are also simple.

The running time is dominated by computing the intersection of at most n lines with an unbounded face of a semi-dynamic arrangement. Using the point set dual to the arrangement (see [2]) and a data structure for maintaining the semi-dynamic convex hull of that point set (see [3]) these intersection points can be computed in $O(\log n)$ time each. See [5] for a detailed description of this technique. This concludes the proof of the following theorem:

Theorem 1 *For every arrangement of n, $n \geq 3$, lines in general position in the plane there exists an inducing polygon P with n edges. P can be constructed in $O(n \log n)$ time.*

3 Conclusions

We presented an algorithm for the construction of an inducing polygon of size n for an arrangement of n straight lines in general position in the plane. Since we only used the fact that every line intersects every other line exactly once, the result also holds for arrangements of pseudo-lines.

We conjecture that in general there are many inducing n-gons for an arrangement of n lines.

Acknowledgments

We thank Helmut Alt for fruitful discussions.

References

[1] P. Bose, H. Everett, and S. Wismath. Properties of arrangement graphs. *Int. J. of Computational Geometry and Applications*, 13(6):447–462, 2003.

[2] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry.* Springer, Berlin, 3rd ed., 2008.

[3] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32(2):249–267, 1992.

[4] E. Mumford, L. Scharf, and M. Scherfenberg. Inducing polygons of line arrangements. In *Proc. 24th European Workshop on Computational Geometry*, 2008.

[5] L. Scharf and M. Scherfenberg. Inducing n-gon of a line arrangement. Technical Report B 08-14, Freie Universität Berlin, 2008.

[6] L. Scharf and M. Scherfenberg. Inducing polygons of line arrangements. In *Proc. 19th Int. Symp. on Algorithms and Computation*, LNCS 5369: 508–520, 2008.

Large bichromatic point sets admit empty monochromatic 4-gons[*]

O. Aichholzer[†] T. Hackl[‡] C. Huemer[§] F. Hurtado[¶] B. Vogtenhuber[‖]

Abstract

We consider a variation of a problem stated by Erdős and Szekeres in 1935 about the existence of a number $f^{\mathrm{ES}}(k)$ such that any set S of at least $f^{\mathrm{ES}}(k)$ points in general position in the plane has a subset of k points that are the vertices of a convex k-gon. In our setting the points of S are colored, and we say that a (not necessarily convex) spanned polygon is monochromatic if all its vertices have the same color. Moreover, a polygon is called empty if it does not contain any points of S in its interior. We show that any bichromatic set of $n \geq 5044$ points in $\mathbb{R}^2$ in general position determines at least one empty, monochromatic quadrilateral (and thus linearly many).

1 Introduction

Throughout this paper all point sets in the plane are assumed to be in *general position*, i.e., no three points in the set are collinear. When a subset T of a point set S is the vertex set of a polygon P, we say that S *contains* P, that T *determines* P, or that P is *spanned* by points in S.

Erdős and Szekeres [6] asked the following question. "What is the smallest integer $f^{\mathrm{ES}}(k)$ such that any set of $f^{\mathrm{ES}}(k)$ points contains at least one convex k-gon?" In the mathematical history this problem is also known as the "Happy End Problem", see e.g. [3, 11]. Exact values have been known for $k \leq 5$ and only very recently the case $k = 6$ has been settled by Szekeres and Peters [17] by an exhaustive computer search. For $k \geq 7$ upper bounds exist, but exact values are unknown. See also Erdős and Guy [5] for the related problem on the smallest number of convex k-gons determined by any set of n points in the plane.

In a variation raised by Erdős the k-gons are required to be empty, i.e., to not contain any point of the point set in its interior. For $k \leq 5$ exact lower bounds on the number of points to guarantee the existence of an empty k-gon are known: As already observed by Esther Klein, every set of 5 points determines an empty convex 4-gon, and 10 points always contain an empty convex pentagon, a fact proved by Harborth [10]. However, Horton showed that there exist arbitrarily large sets of points which do not contain any empty heptagon [12]. Until recently the existence of empty hexagons remained open, but in 2005 Gerken [9] and independently Nicolás [15] proved that every sufficiently large point set contains a convex empty hexagon. Valtr [20] gives a simpler version of Gerken's proof, but requires more points. As for a lower bound it is known that at least 30 points are needed, that is, there exists a set of 29 points without empty convex hexagons [16].

Several variations on the preceding problems when the points in S belong to different classes – that are usually described as *colors* – were introduced by Devillers et al. [4]. In particular, a polygon spanned by points in S is called *monochromatic* if all its vertices have the same color, and it was proved in [4] that any bichromatic set of n points in the plane determines at least $\lceil \frac{n}{4} \rceil - 2$ monochromatic triangles with pairwise disjoint interiors, which is tight. Later it was shown in [1] that any bichromatic set of n points contains at least $\Omega(n^{5/4})$ empty monochromatic triangles (no disjointness is required), and they conjectured that any bichromatic set of n points in $\mathbb{R}^2$ in general position spans a quadratic number of empty monochromatic triangles. For values of k larger than 3, Devillers et al. showed that for $k \geq 5$ and any n there are bichromatic sets of n points where no empty monochromatic convex k-gon exists (Theorem 3.4 in [4]).

It is natural to wonder whether similar results are possible when there are more than two colors. In [4] (Theorem 3.3) this question has been settled by showing that already for three colors there are sets not even spanning any empty monochromatic triangle.

Hence, the interesting remaining case is the existence of empty monochromatic (convex) quadrilaterals in bichromatic point sets. Figure 1 shows a set with 18 points which does not contain an empty monochromatic quadrilateral, and larger examples with 20 [2], 30 [7], 32 [21] and most recently 36 [13] points have been found. However, all these examples

[*]Research was partially carried out during the visit of Clemens Huemer and Ferran Hurtado at the Institute for Software Technology, Graz University of Technology, Austria.

[†]Institute for Software Technology, University of Technology, Graz, Austria, oaich@ist.tugraz.at

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

[§]Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, clemens.huemer@upc.edu

[¶]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, Ferran.Hurtado@upc.edu

[‖]Institute for Software Technology, University of Technology, Graz, Austria, bvogt@ist.tugraz.at

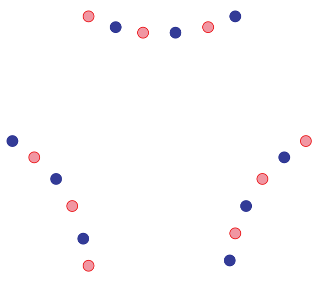


Figure 1: Example without empty monochromatic 4-gons

do contain non-convex empty monochromatic quadrilaterals, while the one in Figure 1 does not.

Notice that every point set that admits an empty convex heptagon will contain an empty monochromatic convex quadrilateral for any bicoloration, because at least four of the vertices of the heptagon will have the same color; however, it is known that for $n \geq 64$ any bichromatic Horton set contains empty monochromatic convex quadrilaterals. These facts led to Conjecture 3.1 in [4], which states that for sufficiently large n any bichromatic set contains at least one empty monochromatic convex quadrilateral.

To date this conjecture has not even been settled for quadrilaterals which are not required to be convex, a weaker conjecture that arose later [8, 14], as no progress in the original formulation had been obtained. In this paper we show that this relaxed version of the conjecture is true: if the cardinality of the bichromatic point set S is sufficiently large, there is always an empty (possibly non-convex) monochromatic quadrilateral spanned by S. To this goal, we prove several sufficient conditions and then show that for large point sets at least one of them must hold.

As already mentioned, throughout this paper we assume S to be a set of n points in the plane in general position, that is, no three points of S lie on a common line. Also, for the sake for brevity, we will use the term 4-gon instead of quadrilateral for the rest of this paper. If $S = R\dot{\cup}B$ is a two-color partition of a point set, then we also write $S = (R, B)$, where R is the set of *red*, and B the set of *blue* points, respectively, with $r = |R|$, $b = |B|$, $r, b \geq 0$ and $n = r + b$.

With $CH(S)$ we denote the set of points of S on the boundary of the convex hull of S (the extreme points of S), with $h = |CH(S)|$ their number and with $i = n - h$ the number of points in the interior of S. Similarly we define $CH(R)$, $h_r = |CH(R)|$, and $i_r = r - h_r$.

2 Preliminaries on uncolored point sets

Let us start with a result on triangulations for (uncolored) point sets which is of interest on its own.

Lemma 1 *Let S be a set of n points in general position in the plane and let π be a fixed parity (even or odd). Then there exists a triangulation $T(S)$ of S such that the parity of the degrees in $T(S)$ of at least $2\lfloor \frac{n-1}{4} \rfloor$ points from S is π.*

Proof. Omitted. □

For odd parity the preceding result can be slightly strengthened to a lower bound of $\frac{n-1}{2}$. As the improvement is marginal and we believe that a much better result is possible, we skip the details and formulate the following problem instead:

Open problem 1 *Which is the maximum value of a constant c such that for any set S of n points in general position there exists a triangulation $T(S)$ in which at least $cn - o(n)$ points of S have odd degree?*

Next we consider a fixed triangulation $T(S)$ of S and give lower bounds of how many triangles of $T(S)$ have to be "pierced" (by placing an obstacle in the interior) so that $T(S)$ does not contain any unpierced (that is, empty) 4-gon. The number $\#_{odd}$ of points of S with odd degree in $T(S)$ plays a central role, as for each interior point with degree δ we need to pierce at least $\lceil \frac{\delta}{2} \rceil$ incident triangles. Summing over all points leads to the following lemma.

Lemma 2 *If for a triangulation $T(S)$ the number of pierced triangles is less than $n + \frac{\#_{odd} - 4h - 6}{6}$, then there exists an unpierced 4-gon in $T(S)$.*

Proof. Omitted. □

3 Bichromatic sets with small convex hulls

Now let $S = (R, B)$ be a bichromatic set. We will triangulate R and use the results of the previous section to get bounds for piercing the resulting red quadrilaterals with blue points from B. From Lemma 2 we immediately get:

Lemma 3 *Let $S = (R, B)$ be a bichromatic point set and $T(R)$ a triangulation of R. With $\#_{odd(R)}$ we denote the number of points of R with odd edge degree in $T(R)$. If $b < r + \frac{\#_{odd(R)} - 4h_r - 6}{6}$ then there exists at least one red empty 4-gon consisting of two adjacent triangles in $T(R)$.*

A consequence of Lemma 3 is a relation between the number of points in R of odd degree in $T(R)$ and the size of the convex hull of R, namely that if $\#_{odd(R)} > 4h_r + 6 - 6(r - b)$ then there exists at least one empty red 4-gon in the triangulation $T(R)$. We can now combine this fact with the choice of an appropriate triangulation $T(R)$, with $\#_{odd(R)} \geq 2\left\lfloor \frac{r-1}{4} \right\rfloor$, whose existence has been proved in Lemma 1, and we get:

Proposition 4 *Let $S = (R, B)$ be a bichromatic point set. If $h_r < \frac{\lfloor \frac{r-1}{4} \rfloor - 3}{2} + \frac{3}{2}(r-b)$, then S contains at least one red empty 4-gon.*

Note that for this result the role of R and B can of course be switched. Proposition 4 also shows that the worst case occurs if R and B have the same cardinality. In this case, or more generally for $r \geq b$, we can simplify the bound to $h_r < \frac{r}{8} - 2$. In particular, this proves that if the convex hull of the larger subset has sub-linear size, we immediately get an empty monochromatic 4-gon.

4 Bichromatic sets with large discrepancy

In this section we consider the case that the cardinalities of the red and the blue set differ significantly. As a first step we generalize a result of Sakai and Urrutia [18] on convex empty, monochromatic 4-gons to simple, but not necessarily convex, 4-gons.

Lemma 5 *If in a bichromatic point set $r \geq \frac{3}{2}b + 4$ then there exists at least one empty red 4-gon in R.*

Proof. The proof is based on induction over b and follows the lines given in [18]. Details are omitted. ☐

Let us recall that for a point set with an even number of extreme points, a quadrangulation is a maximal planar bipartite graph. If the size of the convex hull is odd, we allow one triangle. The number of 4-gons in a quadrangulation of a point set is given in the next observation in terms of n and h, a fact that will be used in Lemma 6.

Observation 1 *A quadrangulation $Q(S)$ on a point set S with n points and h extreme points contains $n - \lceil \frac{h}{2} \rceil - 1$ empty 4-gons.*

Notice that Lemma 5 can be rephrased in the form that $\frac{b}{r-4} > \frac{2}{3}$ is a necessary condition for S to not contain any empty monochromatic 4-gons. In combination with Observation 1 this leads to an interesting iterative relation between the size of a set $S = (R, B)$ not containing an empty monochromatic 4-gon and the maximum discrepancy between R and B.

Lemma 6 *Let $k \geq 4$ be a constant, $f(k) = \frac{4}{3}k(2k-1)$, and $g(k) = \frac{k}{k+2}$. If for every set $S' = (R', B')$ with $|R'| = r' \geq f(k)$, $|B'| = b'$ the inequality*

$$\frac{b'}{r'-4} > g(k) \tag{1}$$

is a necessary condition for S' to not contain an empty monochromatic 4-gon, then for every set $S = (R, B)$ with $r \geq f(k+1)$ the inequality

$$\frac{b}{r-4} > g(k+1) \tag{2}$$

is a necessary condition for S to not contain an empty monochromatic 4-gon.

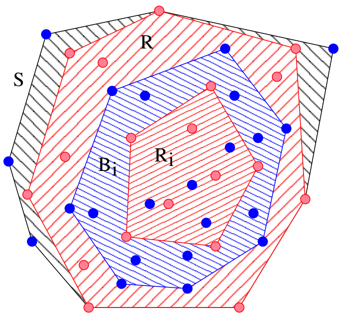


Figure 2: Proof of Lemma 6: red and blue layers.

Proof. The proof is based on the cardinality of nested red and blue convex layers as depicted in Figure 2. Details are omitted. ☐

We are now ready to show that for sets with sufficiently large cardinality the factor of discrepancy has to be arbitrary small in order to avoid empty monochromatic 4-gons.

Proposition 7 *For $l \in \mathbb{N}$, $l \geq 4$, a set $S = (R, B)$ with $r \geq \frac{4}{3}l(2l-1)$ and $b \leq \frac{l}{l+2}(r-4)$ contains an empty monochromatic 4-gon.*

Proof. For $l = 4$ the statement follows directly from Lemma 5. For $l > 4$ we iteratively apply Lemma 6 for $k = 4 \ldots l-1$, where the precondition for the first iteration is given by Lemma 5. Proposition 7 then follows from the result of the last step. ☐

5 Putting things together

As a consequence of Proposition 7 we derive a lower bound on the number of extreme points of the red set, which guarantees the existence of empty 4-gons.

Lemma 8 *For $l \in \mathbb{N}$, $l \geq 4$ let $S = (R, B)$ be a set with $r \geq \frac{4}{3}(l+1)(2l+1)$. Then $h_r \geq r\frac{2(2l+3)}{(l+2)(l+3)} + \frac{8l}{l+3}$ implies that S contains an empty monochromatic 4-gon.*

Proof. The proof is based on Proposition 7 and on the cardinality of nested convex layers, similar to the proof of Lemma 6. Details are omitted. ☐

By combining the results of Proposition 4 and Lemma 8 we finally obtain our main result.

Theorem 9 *Every bichromatic set $S = (R, B)$ with $n \geq 5044$ contains an empty monochromatic 4-gon.*

Proof. Without loss of generality, assume that $r \geq b$. From Proposition 4 we know that $h_r < \frac{\lfloor \frac{r-1}{4} \rfloor - 3}{2} + \frac{3}{2}(r-b)$ is a sufficient condition to obtain an empty monochromatic 4-gon. This can be simplified to $h_r < \frac{r}{8} - 2$. On the other hand, Lemma 8 provides $h_r \geq r\frac{2(2l+3)}{(l+2)(l+3)} + \frac{8l}{l+3}$ for $l \geq 4$ and $r \geq \frac{4}{3}(l+1)(2l+1)$ as a second sufficient condition. So if

$$\frac{r}{8} - 2 \geq r\frac{2(2l+3)}{(l+2)(l+3)} + \frac{8l}{l+3} \qquad (3)$$

holds, then Theorem 9 follows. Using the inequality $r \geq (4/3)(l+1)(2l+1)$ it follows that inequality (3) is fulfilled for $l \geq 30$, and thus for any set with $r \geq 2522$; in other words, for any set with $n \geq 5044$ points. □

6 Open problems

The existence of convex empty monochromatic 4-gons in sufficiently large bichromatic point sets is still open. It seems that the techniques used in our approach cannot be generalized to the convex case, as convexity invalidates several of our lemmas and intermediate results. Another interesting open question are non-convex empty monochromatic k-gons for $k > 4$.

It would also be interesting to establish a 3D version of these results for hexahedra consisting of two tetrahedra sharing a face. Let us recall in this respect that Urrutia [19] proved that in any 4-colored point set in $\mathbb{R}^3$ in general position there is at least one empty monochromatic tetrahedron (in fact, a linear number of them).

Let us finally mention again Open Problem 1, which asks which is the maximum constant c such that for any point set there always exists a triangulation where $cn - o(n)$ points have odd degree.

Acknowledgments

We thank Wolfgang Aigner, Franz Aurenhammer, Bernhard Kornberger, Günter Rote, and Ludmila Scharf for helpful discussions. Research of Oswin Aichholzer, Thomas Hackl, and Birgit Vogtenhuber supported by the FWF [Austrian Fonds zur Förderung der Wissenschaftlichen Forschung] under grant S9205-N12, NFN Industrial Geometry. Research of Clemens Huemer and Ferran Hurtado partially supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

References

[1] O. Aichholzer, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, J. Urrutia, Empty monochromatic triangles. *Proc. 20th Canadian Conference on Computational Geometry*, 20:75–78, Montreal, Quebec, Canada, 2008.

[2] P. Brass, Empty monochromatic fourgons in two-colored points sets. *Geombinatorics*, XIV(1):5–7, 2004.

[3] P. Brass, W. Moser, J. Pach, *Research Problems in Discrete Geometry*, Springer, 2005.

[4] O. Devillers, F. Hurtado, G. Károlyi, C. Seara, Chromatic variants of the Erdős-Szekeres Theorem. *Computational Geometry, Theory and Applications*, 26(3):193–208, 2003.

[5] P. Erdős, R.K. Guy, Crossing number problems. *Amer. Math. Monthly*, 88:52–58, 1973.

[6] P. Erdős, G. Szekeres, A combinatorial problem in geometry. *Compositio Math. 2*, 463–470, 1935.

[7] E. Friedmann, 30 two-colored points with no empty monochromatic convex fourgons. *Geombinatorics*, XIV,(2):53–54, 2004.

[8] F. Hurtado, Open Problem Session. *European Workshop on Computational Geometry*, Eindhoven, The Netherlands, 2005.

[9] T. Gerken, Empty convex hexagons in planar point sets. *Discrete and Computational Geometry*, 39(1-3):239–272, 2008.

[10] H. Harborth, Konvexe Fünfecke in ebenen Punktmengen. *Elem. Math.*, 33:116–118, 1978.

[11] P. Hoffmann, The man who loved only numbers. Hyperion, New York, 1998.

[12] J.D. Horton, Sets with no empty convex 7-gons. *Canad. Math. Bull.*, 26(4):482–484, 1983.

[13] C. Huemer, C. Seara, 36 two-colored points with no empty monochromatic convex fourgons. (Submitted)

[14] J. Pach, On simplices embracing a point (invited talk). *Topological & Geometric Graph Theory (TGGT)*, Paris, France, 2008.

[15] C.M. Nicolás, The empty hexagon theorem. *Discrete and Computational Geometry*, 38(2):389–397, 2007.

[16] M.H. Overmars, Finding sets of points without empty convex 6-gons. *Discrete and Computational Geometry*, 29:153–158, 2003.

[17] G. Szekeres, L. Peters, Computer solution to the 17-point Erdős-Szekeres problem. *The Anziam Journal*, 48(2):151–164, 2006.

[18] T.Sakai, J.Urrutia, Covering the convex quadrilaterals of point sets. *Graphs and Combinatorics*, 23:343–358, 2007.

[19] J. Urrutia, Coloraciones, tetraedralizaciones, y tetraedros vacios en coloraciones de conjuntos de puntos en R^3. *Proc. X Encuentros de geometria Computacional*, Sevilla, 95–100, 2003.

[20] P. Valtr, On empty hexagons. *in: J. E. Goodman, J. Pach, and R. Pollack, Surveys on Discrete and Computational Geometry, Twenty Years Later*, AMS, 433–441, 2008.

[21] R. Van Gulik, 32 two-colored points with no empty monochromatic convex fourgons. *Geombinatorics*, XV,(1):32–33, 2005.

On Plane Geometric Spanners

Prosenjit K. Bose[*]

Abstract

A geometric graph G is a graph whose vertices are points in plane and whose edges are line segments weighted by the Euclidean distance between their endpoints. In this setting, a t-spanner of G is a spanning subgraph G' with the property that for every pair of vertices x, y, the shortest path from x to y in G' has weight at most $t \geq 1$ times the shortest path from x to y in G. The parameter t is commonly referred to as the *spanning ratio*, the *dilation* or the *stretch factor*. In addition to having bounded spanning ratio, it is desireable to build t-spanners that possess other properties, such as bounded degree, low weight, or fault-tolerance to name a few. In this talk, we are particularly interested in planarity. We review various results in the literature on how to build spanners that are planar.

[*]School of Computer Science, Carleton University, Ottawa, Ontario, Canada `jit@scs.carleton.ca`

Lower and upper bounds on the number of empty cylinders and ellipsoids

O. Aichholzer* F. Aurenhammer† O. Devillers‡ T. Hackl* M. Teillaud‡ B. Vogtenhuber*

Abstract

Given a set $\mathcal{S}$ of n points in three dimensions, we study the maximum numbers of quadrics spanned by subsets of points in $\mathcal{S}$ in several ways. Among various results we prove that the number of empty circular cylinders is between $\Omega(n^3)$ and $O(n^4)$ while we have a tight bound $\Theta(n^4)$ for empty ellipsoids. We also take interest in pairs of empty homothetic ellipsoids, with application to the number of combinatorially distinct Delaunay triangulations obtained by orthogonal projections of $\mathcal{S}$ on a two-dimensional plane, which is $\Omega(n^4)$ and $O(n^5)$.

A side result is that the convex hull in d dimensions of a set of n points, where one half lies in a subspace of odd dimension $\delta > \frac{d}{2}$, and the second half is the (multi-dimensional) projection of the first half on another subspace of dimension δ, has complexity only $O\left(n^{\frac{d}{2}-1}\right)$.

1 Introduction

Counting incidences between geometric objects, and analyzing the number of objects spanned by a finite set of points, are topics of frequent interest in computational and combinatorial geometry. Prominent examples are counting the number of faces of many cells in an arrangement of curves [13], determining the number of straight lines touching a given set of spheres [14], bounding the number of empty spheres defined by a finite set of points (i.e., the number of vertices of the Voronoi diagram), and counting the number of plane graphs spanned by a planar set of points [5].

In this paper, we provide combinatorial bounds on the maximum number of 'combinatorially different' quadrics of various types in $\mathbb{R}^3$. Given a set $\mathcal{S}$ of n points in $\mathbb{R}^3$, we consider quadrics having all points of $\mathcal{S}$ on the same side. For ellipsoids or cylinders, inside and outside are clearly defined, and such a quadric will be called *enclosing* or *empty.* We also prove bounds concerning pairs of *homothetic* quadrics. Our original motivation for studying such pairs was the case of circular cylinders, in order to count the number of combinatorially different 2D Delaunay triangulations obtainable by orthogonal projection of $\mathcal{S}$. Such projections are commonly used in 3D surface reconstruction, e.g., for recovering terrains.

Naturally, our bounds depend on the number of degrees of freedom of the considered object class, which is the dimension of a manifold describing the class in any representation, or, in other words, the minimum number of independent real parameters needed to (locally) specify an object of the class. Before describing our results in more detail, let us briefly recall some facts about quadrics.

Types of quadrics

A general quadric is defined by its equation that contains 9 coefficients plus a constant factor, that is, a quadric has 9 degrees of freedom. Given a subset of k points from $\mathcal{S}$ and a family of quadrics having d degrees of freedom, there exists a family of quadrics with $d-k$ degrees of freedom that are passing through these points. Consequently, a quadric (and also an ellipsoid) can be defined by constraining it to pass through 9 given points. We will call two quadrics *combinatorially different* if they have different subsets of $\mathcal{S}$ on their boundaries. Combinatorially different quadrics spanned by $\mathcal{S}$ are the objects of interest in our complexity bounds.

Circular cylinders, which have only 5 degrees of freedom, can be locally specified by 5 points. (In fact, up to six cylinders through these points may exist [8]). It turns out that finding bounds for empty circular cylinders is more involved than for general quadrics. We also study the somehow intermediate case of general (elliptic) cylinders, which have 7 degrees of freedom. A complete classification of quadrics can be found in Dupont et al. [10]. Observe that two quadrics are homothetic if their equations share the same quadratic part. Thus, a second quadric homothetic to a given one has 4 degrees of freedom (3 for translation and one for the homothety factor). For the particular case of cylinders, translations along the cylinder axis are irrelevant, and thus the second cylinder has 3 degrees of freedom. For circular cylinders, being homothetic is the same as being parallel. From this discussion we deduce that the
- number of circular cylinders (not necessarily empty) defined by a set of n points is $\Theta(n^5)$,
- number of pairs of parallel circ. cylinders is $\Theta(n^8)$,
- number of general cylinders is $\Theta(n^7)$,
- number of pairs of homothetic cylinders is $\Theta(n^{10})$,

*IST, Graz University of Technology, oaich@ist.tugraz.at
†ITCS, Graz University of Technology, auren@igi.tugraz.at
‡INRIA, Sophia Antipolis — Méditerranée. www.inria.fr/
Full version is available [4]. The authors acknowledge support of the Partenariat Hubert Curien Amadeus No 13792TD | FR01/2007 "Geometric concepts and CGAL" http://www.inria.fr/sophia/geometrica/collaborations/Amadeus/.

- number of quadrics is $\Theta(n^9)$, and
- number of pairs of homothetic quadrics is $\Theta(n^{13})$.

Results and related work

This paper provides improved asymptotic upper and lower bounds on the number of combinatorially different quadrics defined by a set of n points in $\mathbb{R}^3$. Upper and lower bounds match in the case of general quadrics, $\Theta(n^4)$, and pairs of them, $\Theta(n^6)$. Surprisingly, both bounds are also valid for general cylinders. Thus, despite the fact that general cylinders have 2 degrees of freedom less than quadrics, the intrinsic complexity does not decrease. Gaps remain in our results for circular cylinders, $\Omega(n^3)$ and $O(n^4)$, and for pairs of parallel circular cylinders, $\Omega(n^4)$ and $O(n^5)$.

The last mentioned result carries over to the number of combinatorially different 2D Delaunay triangulations obtainable by projecting a given point set in $\mathbb{R}^3$. To our knowledge, no nontrivial bounds have been known on this problem, which can be viewed as a three-dimensional generalization of the problem of finding the different orderings of the projection of a set of points. The latter problem is well known as finding circular sequences [11] and has a complexity of $\Theta(n^2)$.

Lower bounds for maximizing the number of certain types of quadrics are derived from a generic construction and its variants. Upper bounds are obtained by applying tailored linearization schemes, introducing a space of quadrics in 9 dimensions, and a space of homothetic quadrics in 13 dimensions (that can be reduced to dimension 12 for cylinders). These spaces are similar to well-known notions such as the spaces of circles, spheres, or conics [11].

Our second linearization scheme becomes powerful in combination with a new theorem (of separate interest) that reduces the complexity of the convex hull of a set of n points in d dimensions for certain special configurations. Namely, if $\frac{n}{2}$ of the points lie in a subspace of dimension $\delta > \frac{d}{2}$, and the second half is the (multi-dimensional) projection of the first half on another subspace of dimension δ, then the complexity is $O\left(n^{\lfloor\frac{\delta}{2}\rfloor+\lfloor\frac{d-\delta}{2}\rfloor}\right)$. This reduces the general upper bound of $O\left(n^{\lfloor\frac{d}{2}\rfloor}\right)$ to $O\left(n^{\frac{d}{2}-1}\right)$ if d is even and δ is odd. The particular case $\delta = d-1$ was already proven [7].

Related work mostly concerns circular cylinders, a kind of quadric useful, for example, in metrology [2, 9] for quality measure of mechanical devices, or in 3D modeling in order to fit special surfaces in a given data set (e.g., trying to model pipes in a factory). Agarwal et al. [3] compute the set of empty unit radius cylinders in $O(n^{3+\varepsilon})$ time and show an $\Omega(n^2)$ complexity bound. Devillers [7] gives $O(n^4)$ and $\Omega(n^3)$ bounds for the number of coaxial circular cylinders defined by 6 points with all points between the two cylinders. There is special interest in smallest enclosing cylinders. Agarwal et al. [3] propose a construction algorithm with near cubic time complexity. Chan [6] proposes a faster approximation algorithm, and Schömer et al. [12] give an algorithm with bit-complexity analysis.

2 Upper bounds

2.1 Empty and enclosing ellipsoids or cylinders

Lemma 1 *The number of combinatorially different* empty *(or* enclosing*) ellipsoids or cylinders defined by a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ is $O(n^4)$.*

Proof. We use a classical linearization scheme similar to the one used in the next section, □

2.2 Pairs of empty homothetic ellipsoids

Lemma 2 *The number of combinatorially different pairs of* empty homothetic *ellipsoids or cylinders defined by a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ is $O(n^6)$.*

Proof. The linearization scheme here is similar to the one used in [1, 7]. To a given point $p = (x_p, y_p, z_p) \in \mathbb{R}^3$ we associate not one but two points in $\mathbb{R}^{13}$:

$p^\star = (x_p^2, y_p^2, z_p^2, x_py_p, x_pz_p, y_pz_p, x_p, y_p, z_p, 0, 0, 0, 0) \in \mathbb{R}^{13}$

$p^\dagger = (x_p^2, y_p^2, z_p^2, x_py_p, x_pz_p, y_pz_p, 0, 0, 0, x_p, y_p, z_p, 1) \in \mathbb{R}^{13}$.

Given two homothetic quadrics Q_ϕ and Q'_ϕ with equations

$\phi_1x^2+\phi_2y^2+\phi_3z^2+\phi_4xy+\phi_5xz+\phi_6yz+\phi_7x+\phi_8y+\phi_9z = \phi_0$

$\phi_1x^2+\phi_2y^2+\phi_3z^2+\phi_4xy+\phi_5xz+\phi_6yz+\phi'_7x+\phi'_8y+\phi'_9z = \phi'_0$,

we define a hyperplane

$$H_{\phi,\phi'} : \phi_1\chi_1+\phi_2\chi_2+\phi_3\chi_3+\phi_4\chi_4+\phi_5\chi_5+\phi_6\chi_6+\phi_7\chi_7+\phi_8\chi_8 +\phi_9\chi_9 + \phi'_7\chi_{10} + \phi'_8\chi_{11} + \phi'_9\chi_{12} + (\phi_0 - \phi'_0)\chi_{13} = \phi_0$$

in dimension 13 (where χ_i are the coordinates in $\mathbb{R}^{13}$). Now we have

$$\begin{aligned} p \in Q_\phi &\iff p^\star \in H_{\phi,\phi'} \\ \text{and} \quad p \in Q_{\phi'} &\iff p^\dagger \in H_{\phi,\phi'}. \end{aligned}$$

The following two statements are equivalent: (1) An ellipsoid Q_ϕ passes through points $p_1, p_2, \ldots, p_j \in \mathcal{S}$, and a ellipsoid $Q_{\phi'}$ homothetic to Q_ϕ passes through $p_{j+1}, p_{j+2}, \ldots, p_k \in \mathcal{S}$, and both Q_ϕ and $Q_{\phi'}$ are empty of other points of $\mathcal{S}$. (2) The corresponding hyperplane $H_{\phi,\phi'}$ passes through $p_1^\star, p_2^\star, \ldots, p_j^\star, p_{j+1}^\dagger, \ldots, p_k^\dagger \in \mathcal{S}^{\star\dagger} = \{p^\star, p^\dagger \mid p \in \mathcal{S}\}$ and all other points of $\mathcal{S}^{\star\dagger}$ are above $H_{\phi,\phi'}$.

Thus pairs of empty ellipsoids correspond to supporting hyperplanes of the convex hull of $\mathcal{S}^{\star\dagger}$, which has a complexity of $O\left(n^{\lfloor\frac{d}{2}\rfloor}\right)$, where $d = 13$ is the dimension of the underlying space, that is, a complexity of $O(n^6)$. □

2.3 Pairs of empty homothetic cylinders

Lemma 3 *The number of combinatorially different pairs of* empty homothetic *cylinders defined by a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ is $O(n^5)$.*

Proof. Assume, without loss of generality, that there is no horizontal cylinder. The second cylinder can be defined from the first one by a homothety and a horizontal translation. It yields to two homothetic cylinders Q_ϕ and Q'_ϕ whose equations verify $\phi_9 = \phi'_9$ which allows to reduce the dimension of the linearization space by one, going down to $\mathbb{R}^{12}$.

Now we are dealing with the convex hulls of points $p^\star$ belonging to a 9 dimensional space and points $p^\dagger$ which are the projection of the points $p^\star$ on another 9 dimensional space parallel to a 3 dimensional vector space. Thus we can apply our general projection theorem (Theorem 4 in the following section) and get an upper bound of $O\left(n^{\lfloor\frac{9}{2}\rfloor+\lfloor\frac{3}{2}\rfloor}\right) = O(n^5)$. ☐

3 General projection theorem

For special configurations of points in d dimensions, the complexity of the convex hull cannot reach the worst case of $\Theta\left(n^{\lfloor\frac{d}{2}\rfloor}\right)$. For example, the projection theorem in [7] proves that for a point set $\mathcal{V}$ of size $2n$, constructed by taking n points in a hyperplane and their parallel projections on another hyperplane, the complexity of its convex hull $CH(\mathcal{V})$ reduces to $O\left(n^{\lfloor\frac{d-1}{2}\rfloor}\right)$. This is of interest if d is even.

A multi-dimensional projection is defined by a vector space $\vec{V}$ of dimension $d-\delta$. Given a point p, its projection p' on Π' is the intersection of Π' and the affine subspace parallel to $\vec{V}$ passing through p; this projection is a unique point if $\vec{V}$ is supplementary to the direction of Π' (no $\vec{v} \in \vec{V}$ is parallel to Π').

We generalize the projection theorem to the case where the points in $\mathbb{R}^d$ live in two δ-dimensional affine subspaces, with $\frac{d}{2} < \delta \le d-1$. If d is even and δ is odd, we save a linear factor like in the original version of the projection theorem. More precisely, we have:

Theorem 4 (General projection theorem) *For any constant dimension d, let Π, Π' be two δ-dimensional affine subspaces in $\mathbb{R}^d$, with $\delta > \frac{d}{2}$, and let $\vec{V}$ be a $(d-\delta)$-dimensional vector space of $\mathbb{R}^d$ supplementary to the directions of both Π and Π'. Let $\mathcal{U} \subset \Pi$ be a set of n points, such that $CH(\mathcal{U}) \cap \Pi' = \emptyset$, and let $\mathcal{U}' \subset \Pi'$ be the projection parallel to $\vec{V}$ of $\mathcal{U}$ on Π'. Let $\mathcal{V} = \mathcal{U} \cup \mathcal{U}'$. The convex hull $CH(\mathcal{V})$ of $\mathcal{V}$ has asymptotic complexity $O\left(n^{\lfloor\frac{\delta}{2}\rfloor+\lfloor\frac{d-\delta}{2}\rfloor}\right)$. If d is even and δ is odd, the complexity becomes $O\left(n^{\frac{d}{2}-1}\right)$.*

Proof. (Sketch) (see [4] for details). A full dimensional face of $CH(\mathcal{V})$ combines a face of $CH(\mathcal{U})$ with a face of $CH(\mathcal{U}')$. The proof then goes in two steps: first a facet of $CH(\mathcal{V})$ contains a $(\delta-1)$-dimensional face that projects on a facet of $CH(\mathcal{U})$ (and there is a constant number of such faces for each facet of $CH(\mathcal{U})$); second the number of facets of $CH(\mathcal{V})$ containing a given $(\delta-1)$-dimensional face has the complexity of a convex hull in dimension $d-\delta$. Combining these two points gives the claimed complexity. ☐

4 Lower bounds

We now turn our attention to lower bound constructions. We exhibit families of points lying on a few straight lines that allow for a large number of empty (or enclosing) quadrics of several types. By slightly perturbing these sets we can achieve general position without altering the number of objects counted.

4.1 Empty circular cylinders

Lemma 5 *There exists a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ such that the number of combinatorially different empty circular cylinders defined by $\mathcal{S}$ is $\Omega(n^3)$.*

Proof. Consider the points $p_0 = (1,0,0)$ and $q_0 = (0,1,0)$. Then the ellipses in the plane $z = 0$ and tangent to the x axis at p_0 and to the y axis at q_0 form a continuous family that can be parameterized by $\lambda \in [1,\infty[$, the ratio of the long axis to the small axis (Figure 1a). Denote with E_λ one of these ellipses and consider the ellipses $E_{1+\frac{k}{n}}$ for $0 \le k < N$ for $N = \frac{n}{3}$.

Through each of these ellipses, we can fit one circular cylinder such that the vector of the direction of the cylinder axis is positive in all three coordinates (there is another one with only x and y values negative).

Now, since the slopes of those cylinders are different, we get disjoint ellipses if we consider a cross section of the cylinders by a plane $z = h$ for large enough h. Thus we can add points r_k on the line $x - y = z - h = 0$ that separate these ellipses as indicated in Figure 1b. By slightly tilting the cylinders until for every k the cylinder corresponding to $E_{1+\frac{k}{n}}$ touches r_k, we create a linear size family of cylinders through p_0, q_0 and r_k which are tangent to the x and y axes.

Considering points $p_i = (1+\frac{i\varepsilon}{n}, 0, 0)$ and $q_j = (0, 1+\frac{j\varepsilon}{n}, 0)$, $0 \le i, j < N$, for small enough ε performs a small perturbation on the ellipses and so does neither change the existence of the cylinders nor the fact that they are empty, since all cylinders are tangent to the x and y axes which contain the p_i and q_j points (Figure 1cd). This construction produces empty circular cylinders through p_i, q_j, r_k for all i, j, k. ☐

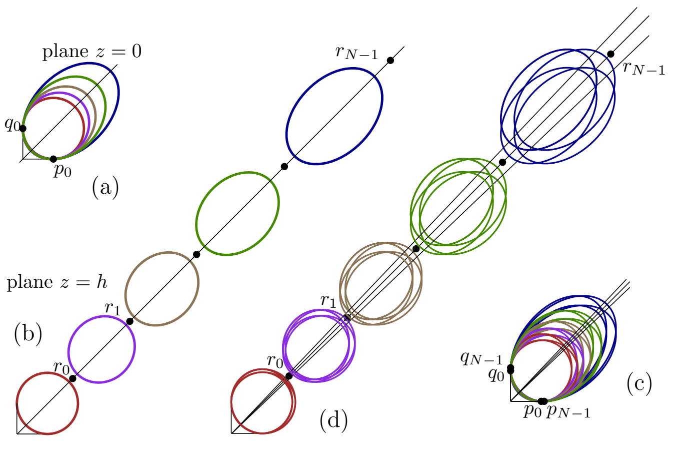


Figure 1: (ab) Cross sections of cylinders through p_0 and q_0 and tangent to x and y axes. (cd) Cross sections of cylinders through p_i and q_j and tangent to x and y axes.

4.2 Empty pairs of parallel circular cylinders

Lemma 6 *There exists a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ such that the number of combinatorially different pairs of empty parallel circular cylinders defined by $\mathcal{S}$ is $\Omega(n^4)$.*

Proof. Let now $N = \frac{n}{4}$. We add a family of points to the construction in the proof of Lemma 5, namely $s_l = (L + \frac{l\varepsilon}{n}, 0, 0)$, $0 \leq l < N$, and a single point $u = (L-1, 1, 0)$, for L large enough. For each of the $\Omega(n^3)$ cylinders through p_i, q_j, r_k for all $0 \leq i, j, k < N$ we can place a disjoint parallel circular cylinder through u and any of the points s_l and tangent to the x axis, avoiding all the points p_i, q_j, r_k. □

4.3 Empty general cylinders (and pairs of them)

Lemma 7 *There exists a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ such that the number of combinatorially different empty general cylinders defined by $\mathcal{S}$ is $\Omega(n^4)$. Moreover, the number of combinatorially different pairs of empty homothetic general cylinders defined by $\mathcal{S}$ is $\Omega(n^5)$.*

Proof. Omitted [4] □

4.4 Empty ellipsoids (and pairs of them)

Lemma 8 *There exists a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ such that the number of combinatorially different empty ellipsoids defined by $\mathcal{S}$ is $\Omega(n^4)$. Moreover, the number of combinatorially different pairs of empty homothetic ellipsoids defined by $\mathcal{S}$ is $\Omega(n^6)$.*

Proof. Omitted [4] □

4.5 Enclosing ellipsoids or cylinders

Lemma 9 *There exists a set $\mathcal{S}$ of n points in $\mathbb{R}^3$ such that the number of combinatorially different enclosing ellipsoids defined by $\mathcal{S}$ is $\Omega(n^4)$, and a set such that the number of combinatorially different enclosing cylinders defined by $\mathcal{S}$ is $\Omega(n^3)$.*

Proof. Omitted [4] □

5 2D Delaunay triangulations from projection

Corollary 10 *Given a set of n points in $\mathbb{R}^3$, the number of combinatorially different 2D Delaunay triangulations obtainable by projecting these points on a plane is $O(n^5)$. Moreover, there exist sets of n points in $\mathbb{R}^3$ such that this number is $\Omega(n^4)$.*

Proof. (Sketch) If a direction of projection is represented by a point on $\mathbb{S}_2$, there exists a map on $\mathbb{S}_2$ whose cells contain directions, giving the same Delaunay triangulation. Vertices of this map correspond to pairs of parallel empty circular cylinders. □

References

[1] Agarwal, Aronov, Koltun, & Sharir. Lines avoiding unit balls in three dimensions. *DCG*, 34:231–250, 2005.

[2] Agarwal, Aronov, & Sharir. Line traversals of balls and smallest enclosing cylinders in three dimensions. In *SODA*, 483–492, 1997.

[3] Agarwal, Aronov, & Sharir. Line traversals of balls and smallest enclosing cylinders in three dimensions. *DCG*, 21:373–388, 1999.

[4] Aichholzer, Aurenhammer, Devillers, Hackl, Teillaud, & Vogtenhuber. Counting quadrics and Delaunay triangulations and a new convex hull theorem. Report 6748, INRIA, 2008. `http://hal.inria.fr/inria-00343651`

[5] Aichholzer, Hackl, Huemer, Hurtado, Krasser, & Vogtenhuber. On the number of plane geometric graphs. *Graphs and Combinatorics*, 23:67–84, 2007.

[6] Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *SoCG*, 300–309, 2000.

[7] Devillers. The number of cylindrical shells. *DCG*, 30:453–458, 2003.

[8] Devillers, Mourrain, Preparata, & Trebuchet. Circular cylinders by four or five points in space. *DCG*, 29:83–104, 2003.

[9] Devillers & Preparata. Evaluating the cylindricity of a nominally cylindrical point set. In *SODA*, 518–527, 2000.

[10] Dupont, Lazard, Lazard, & Petitjean. Near-optimal parameterization of the intersection of quadrics: I. The generic algorithm. *JoSC*, 168–191, 2008.

[11] Edelsbrunner. *Algorithms in Combinatorial Geometry*, Springer, 1987.

[12] Schömer, Sellen, Teichmann, & Yap. Smallest enclosing cylinders. *Algorithmica*, 27(2):170–186, 2000.

[13] Sharir & Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, 1995.

[14] Sottile & Theobald. Lines tangent to $2n-2$ spheres in R^n. *Trans. Amer. Math. Soc.*, 354:4815–4829, 2002.

Efficient Enumeration of All Pseudoline Arrangements

Katsuhisa Yamanaka[*] Shin-ichi Nakano[†] Yasuko Matsui[‡] Ryuhei Uehara[§] Kento Nakada[¶]

Abstract

In this paper we give an algorithm to enumerate all arrangements of n pseudolines. After $O(n^2)$ time preprocessing, our algorithm enumerates each arrangement in $O(1)$ time for each and uses $O(n^2)$ space.

1 Introduction

Arrangements of lines and pseudolines are one of important and appealing objects in the area of geometry and combinatorics.

Felsner [2] worked on the number of arrangements of pseudolines and showed that the number of aranges-ments of n pseudolines is bounded by $2^{0.6974n^2}$. Arrangements of pseudolines are strongly related to oriented matroids, and there is a bijection between arrangements and oriented matroids of rank 3. Several results on enumeration of oriented matroids are known [3, 4].

In this paper we give an efficient algorithm to enumerate all arrangements of n pseudolines.

In [8, 9] we have designed an algorithm to enumerate every "ladder lottery," which is a network of swaps as depicted in Figure 1(c), for a given permutation. For the fixed permutation $\pi = (n, n-1, \dots, 1)$ we can observe that each ladder lottery for π can be regarded as an arrangement of n pseudolines. See Figure 1(a) and (c). In this paper by simplifying the algorithm in [8, 9] to only work for the fixed permutation $\pi = (n, n-1, \dots, 1)$ we design a simple but efficient algorithm to enumerate all arrangements of n pseudolines. After $O(n^2)$ time preprocessing, our algorithm computes each arrangement in $O(1)$ time for each.

The idea of our enumeration algorithm is as follows. Let S_n be the set of all combinatorial stuctures of arrangements of n pseudolines. We first define a tree structure T_n, called *the family tree*, among S_n, (see Figure 2) in which each vertex of T_n corresponds to a

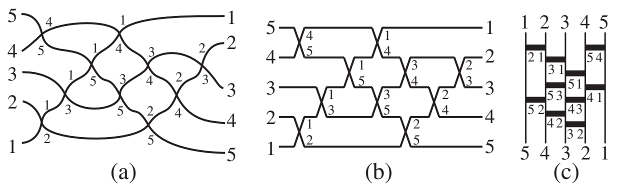


Figure 1: (a) An arrangement of 5 pseudolines, (b) its wiring diagram and (b) the corresponding optimal ladder lottery.

combinatorial structure of an arrangement in S_n and each edge of T_n corresponds to a relation between two combinatorial structures of arrangements which can be transformed to the other by one *local swap operation*, as shown in Figure 3. Then we design an efficient algorithm to generate all child vertices of a given vertex in T_n. Applying the algorithm recursively from the root of T_n, we can generate all vertices in T_n, and also corresponding all combinatorial structures of arrangements in S_n. Based on such a tree structure but with some other ideas a lot of efficient enumeration algorithms are designed [1, 6].

2 Preliminary

A *pseudoline* is an x-monotone curve in the Euclidean plane. An *arrangement* of pseudolines is a set of pseudolines in which every pair intersects exactly once. See Figure 1(a). An arrangement is *simple* if no three pseudolines share a common point. Throughout this paper, the term arrangement always denotes a simple arrangement of pseudolines.

A *wiring diagram*, introduced in [5], of an arrangement of n pseudolines is a network with n lines and $\binom{n}{2}$ intersections. See Figure 1(b). The left ends correspond to the reverse permutation $(n, n-1, \dots, 1)$ in top to bottom order. The right ends correspond to the identical permutation $(1, 2, \dots, n)$ in top to bottom order. Each line i starts at the i-th left end from the bottom, then goes right, however at every intersection the line goes up or down to cross other line, then finally line i reaches the i-th right end from the top. Each such path corresponds to a pseudoline. Note that each path has exactly $n-1$ intersections.

The combinatorial structure of each pseudoline arrangement can be modeled as a wiring diagram, and each wiring diagram models the combinatorial structure of a set of pseudolines arrangements. Note that applying some perturbation to a pseudoline arrange-

[*]Graduate School of Information Systems, The University of Electro-Communications, `yamanaka@is.uec.ac.jp`

[†]Graduate School of Computer Science, Gunma University, `nakano@cs.gunma-u.ac.jp`

[‡]Department of Mathematical Sciences, Tokai University, `yasuko@ss.u-tokai.ac.jp`

[§]School of Information Science, Japan Advanced Institute of Science and Technology, `uehara@jaist.ac.jp`

[¶]Research Institute for Mathematical Sciences, Kyoto University, Partially supported by GCOE, Kyoto University, `nakada@kurims.kyoto-u.ac.jp`

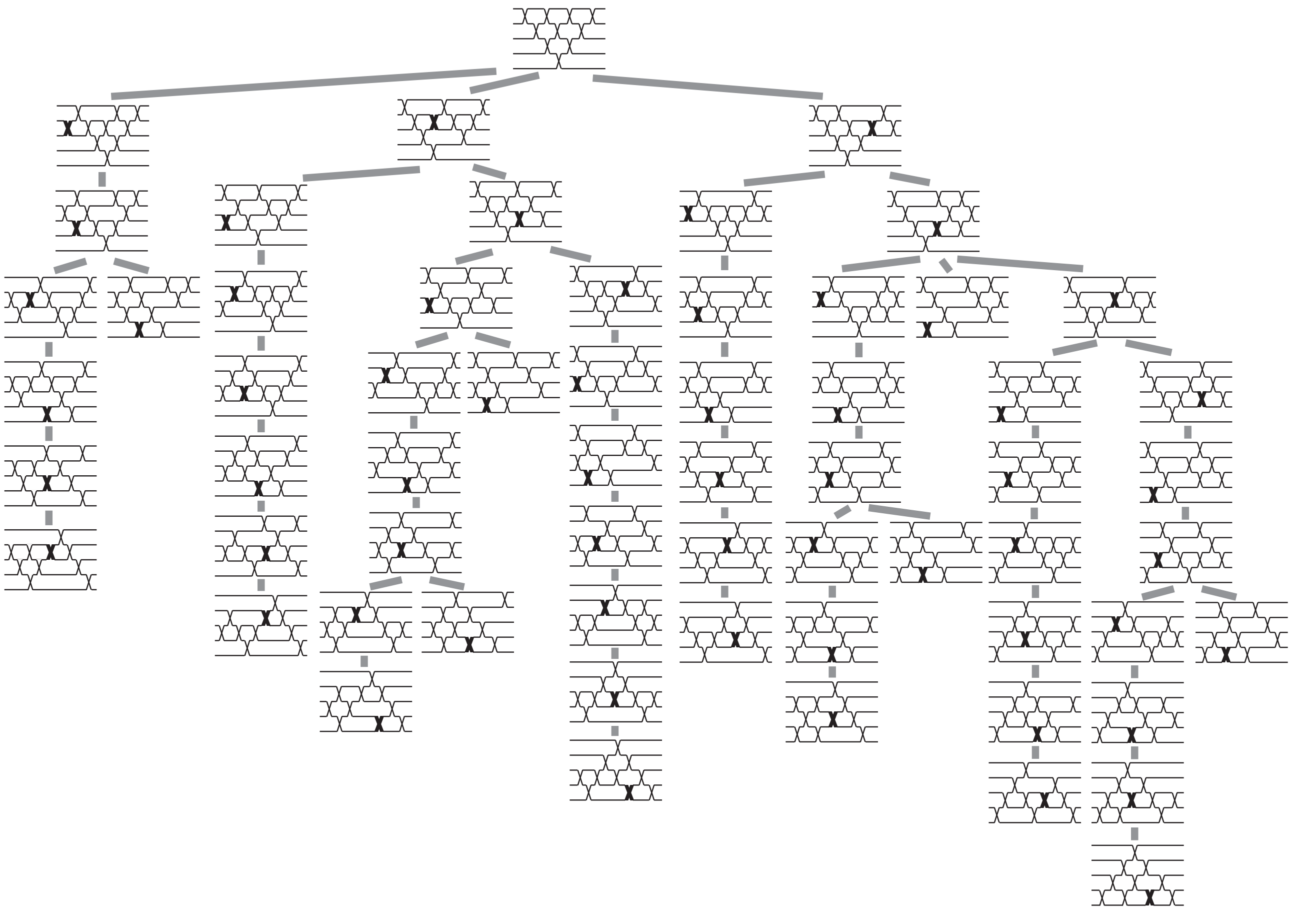


Figure 2: The family tree T_5.

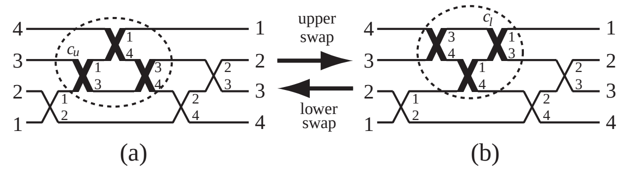


Figure 3: A local swap operation.

ment still results in the same corresponding wiring diagram. We say two pseudoline arrangements are *isomorphic* if there is a bijection between their faces of corresponding wiring diagrams preserving their neighbor relation. In this paper we enumerate all combinatorial structures of pseudoline arrangements by enumerating all distinct wiring diagrams.

A *local swap operation* is a local modification of a wiring diagram, as shown in Figure 3. Note that the dashed circle contains exactly three intersections. Also note that applying this modification to a wiring diagram of a pseudline arrangement results in other wiring diagram of other pseudoline arrangement, since only the "location" of the three intersections has changed. A local swap operation (a) to (b) in Figure 3 is called an *upper swap operation to intersection* c_u. Similarly, a local swap operation (b) to (a) in Figure 3 is called a *lower swap operation to intersection* c_l.

3 The Family Tree

In this section we design a tree structure T_n among wiring diagrams. See Figure 2.

Let S_n be the set of all wiring diagrams corresponding to the set of all combinatorial structures of arrangements of n pseudolines, and $A = A_n$ be an arrangement of n pseudolines. The line in the wiring diagram starting at the i-th left end from the bottom is called *line* i. The line n starts the uppermost left end, then crosses each of other lines exactly once, finally reaches the lowermost right end. Thus the line n contains exactly $n-1$ "downward" intersection, and line n is y-monotone in the wiring diagram. Note that this property does not hold for other line $i \neq n$. See Figure 1(b).

The pseudoline of n partitions A_n into the left part A_n^L and the right part A_n^R. Removing the pseudoline of n from A_n results in an arrangement A_{n-1} of the remaining $n-1$ pseudolines. We say the wiring diagram of A_n is n-*clean* if A_n^L has no intersection. If the wiring diagram of A_n is n-clean then the line $n-1$ is also y-monotone in the wiring diagram of A_{n-1}, and we can define A_{n-2} similarly, and we say the wiring diagram of A_{n-1} is $(n-1)$-clean if A_{n-1}^L has no intersection. We repeat this process until some non-clean wiring diagram appears or the arrangement becomes

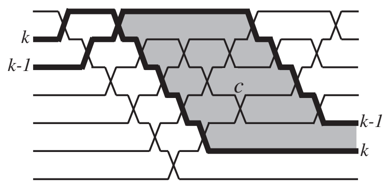


Figure 4: An active region.

empty. If the wiring diagram of A_k is k-clean for each $k = n, n-1, \ldots, 1$, then the wiring diagram of A is called *the root*, denoted by R. See the rightmost diagram in Figure 5. Otherwise, there exists some k such that the wiring diagram of A_i is i-clean for each $i = n, n-1, \ldots, k$, and the wiring diagram of A_{k-1} is not $(k-1)$-clean. We say *the clean level of the wiring diagram of A is k*. Especially if A_n^L has an intersection, then the wiring diagram of A has the clean level $n+1$, and the root R has the clean level 1. Note that if the clean level is k then the lines $n, n-1, \ldots, k$ form so called "brick structure" in the wiring diagram, in which each line $i \geq k-1$ first goes up $n-i$ times, crossing the lines $n, n-1, \ldots, i+1$, in this order, then pass through an uppermost horizontal segment, then go down $i-1$ times. Note that "the region" below line $i \geq k$ contains no intersection of two lines both less than k. Also the region above the line k and below the line $k-1$ contains at least one intersection of two lines both less than k. See Figure 4. The region is called *the active region* of the wiring diagram of A. Especially, we define the active region of R is ϕ for convenience (in the proof of Lemma 2).

Now we assign a wiring diagram in S_n for each wiring diagram W in $S_n \setminus \{R\}$ as follows. Assume that W has the clean level k. Thus the active region of W has at least one intersection. We say an intersection c in the active region is *visible from line $k-1$* if the two lines, say i and j, crossing at c, both next cross to line $k-1$. Among the visible intersections from line $k-1$, the lowermost intersection is called *the active intersection* of W. In Figure 4, intersection c is the active intersection. Applying an upper swap operation to the active intersection of $W \in S_n \setminus \{R\}$ results in other wiring diagram, denoted by $P(W)$, in S_n. We say $P(W)$ is *the parent* of W, and W is a *child* of $P(W)$. Note that the parent of W is unique, while $P(W)$ may have many children. Also note that the clean level of $P(W)$ is smaller or equal to W, and $P(W)$ has less intersections in the active region of W.

We have the following lemma.

Lemma 1 *For any $W \in S_n \setminus \{R\}$, $P(W) \in S_n$ holds.*

Given a wiring diagram W in $S_n \setminus \{R\}$, by repeatedly finding the parent of the derived wiring diagram, we can have the unique sequence $W, P(W), P(P(W)), \ldots$ of wiring diagrams in S_n,

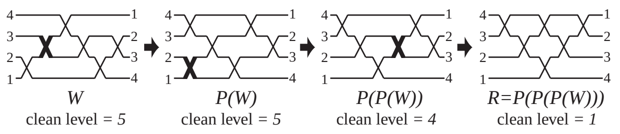


Figure 5: The sequence of a wiring diagram W.

which eventually ends up with the root R in S_n. See Figure 5. The active intersections are depicted by thick lines.

We have the following lemma.

Lemma 2 *The sequence $W, P(W), P(P(W)), \ldots$ of $W \in S_n \setminus \{R\}$ ends with $R \in S_n$.*

Proof. For each $W \in S_n$ we define *the clean potential* $C(W) = (s, t)$, where s is the clean level of W and t is the number of intersections in the active region of W. For $W_1, W_2 \in S_n$ with $C(W_1) = (s_1, t_1)$ and $C(W_2) = (s_2, t_2)$, we say W_1 *is cleaner than* W_2 if (1) $s_1 < s_2$ or (2) $s_1 = s_2$ and $t_1 < t_2$. For any $W \in S_\pi$ we can observe $P(W)$ is cleaner than W. Now R is the cleanest among S_n since $C(R) = (1, 0)$. Thus for any $W \in S_n$ the sequence of clean potentials $C(W), C(P(W)), C(P(P(W))), \ldots$ always ends at $C(R)$. □

By merging all these sequences we can have *the family tree* of S_n, denoted by T_n, in which the root of T_n corresponds to R, the vertices of T_n correspond to the wiring diagrams in S_n and each edge corresponds to a relation between a wiring diagram in S_n and its parent. See Figure 2.

4 Enumerating

In this section we give an efficient algorithm to enumerate all wiring diagrams in S_n.

If we have an algorithm to enumerate all children of a given wiring diagram in S_n, then by recursively applying the algorithm starting at the root R of S_n, we can enumerate all wiring diagrams in S_n, and all corresponding combinatorial structures of arrangements of n pseudolines. Now we design such an algorithm.

We need some definitions. Let $W \neq R$ be a wiring diagram in S_n. Assume W has the clean level k. So each intersection below line $i \geq k$ contains no intersection of two lines both less than k, but in the active region (see Figure 4) there is at least one intersection of two lines, say x and y, with $x, y < k-1$. Each line $i \geq k-1$ goes up $n-i$ times, "turns" at the top, then goes down $i-1$ times. For each line $i = n-1, n-2, \ldots, k-1$, if c is the first intersection to go down after intersections to go up, then c is called *the turn intersection* of line i. Note that line n contains only intersections to go down, so has no turn intersection. Also note that the turn intersection is defined only for line $i = n-1, n-2, \ldots, k-1$ since otherwise it may have many "turns."

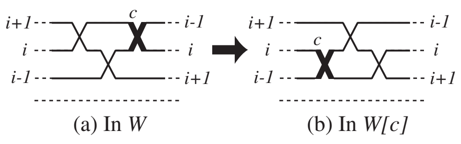


Figure 6: Illustration for Type 1.

Lemma 3 *Let W be a wiring diagram with the clean level k. Every turn intersection of line $i = n-1, n-2, \ldots, k$ can be lower swapped, however any other intersection on line $i = n-1, n-2, \ldots, k$ cannot.*

Proof. An intersection c_l can be lower swapped only if the dashed circle in Figure 3(b) contains exactly three intersections. Because of the brick structure, other lower swaps are interfered by a fourth intersection. □

Let $W[c]$ be the wiring diagram derived from W by applying a lower swap operation to an intersection c. Every child of W is $W[c]$ for some c, but not all $W[c]$ are children of W. $W[c]$ is a child of W only if c is the active intersection of $W[c]$. Now we classify each $W[c]$ as a child of W or not as follows. Remember the clean level of W is k. Let $U(i)$ be the region above line i, and $L(i)$ be the region below line i.

Type 1: c is a turn intersection.

Assume the local structure of W is as shown in Figure 6(a) and c is the turn intersection of line i. If $n-1 \geq i \geq k$ then c can be lower swapped by Lemma 3, and the clean level of $W[c]$ is $i+2$ since $W[c]$ is not $(i+1)$-clear. Thus c is the only intersection in the active region of $W[c]$, so c is the active intersection of $W[c]$. Thus $W[c]$ is a child of W. If $i = k-1$, $W[c]$ is a child of W only when c can be lower swapped.

Type 2: c is not a turn intersection.

We need to consider only intersections which can be lower swapped. Such intersections exist only in $U(k)$. If c is "rightward" visible from neither line k nor $k-1$, then c is not the active intersection in $W[c]$, thus $W[c]$ is not a child of W. So assume otherwise.

If the lower swap operation moves c to $L(k)$, crossing the line k, then the clean level of $W[c]$ is $k+1$ and c is the only intersection in the new active region, so c is the active intersection of $W[c]$. Thus $W[c]$ is a child of W.

If the lower swap operation moves c to $L(k-1)$, crossing the line $k-1$, then the clean level of $W[c]$ remains k, and c is appended to the active region. $W[c]$ is a child of W if c is the active intersection of $W[c]$. Otherwise, $W[c]$ is not a child of W.

By maintaining (i) the clean level k (the clean level of $W[c]$ is always larger than or equal to the clean level of W) and (ii) the list of rightward visible intersections from line k or $k-1$, (those are candidate to be lower swapped, crossing the line k or $k-1$), and (iii) the list of the turn intersections, we can enumerate all children of W in $O(1)$ time for each on average.

Lemma 4 *All children of W can be enumerated in $O(1)$ time for each on average.*

The root R in S_n can be generated in $O(n^2)$ time because of its complete brick structure.

Theorem 5 *After generating and outputting the root R in S_n in $O(n^2)$ time, our algorithm enumerates all combinatorial structures of arrangements of n pseudolines in $O(1)$ time for each on average. The algorithm uses $O(n^2)$ working space.*

By the theorem above, our algorithm generates each wiring diagram in S_n in $O(1)$ time "on average." Using the technique of "prepostordering" [6], one can improve the running time to $O(1)$ time for each in worst case. See [6] for further details of this method: in [6] the method was not explicitly named, and the name "prepostorder" is given by Knuth [7].

Theorem 6 *After $O(n^2)$ time preprocessing, we can enumerate all combinatorial structures of arrangements of n pseudolines in $O(1)$ time for each.*

References

[1] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65(1-3):21–46, 1996.

[2] S. Felsner. On the number of arrangements of pseudolines. *Proc. The 12th annual Symposium on Computational geometry, (SCG 1996)*, 33–37, 1996.

[3] J. Ferté, V. Pilaud, and M. Pocchiola. On the number of simple arrangements of five double pseudolines. In *Proc. Fall Workshop on Computational Geometry, (FWCG 2008)*, 45–46, 2008.

[4] L. Finschi and K. Fukuda. Generation of oriented matroids – a graph theoretical approach. *Discrete Comput. Geom.*, 27:117–136, 2002.

[5] J.E. Goodman. Proof of a conjecture of burr, grünbaum and sloane. *Discrete Math.*, 32:27–35, 1980.

[6] S. Nakano and T. Uno. Constant time generation of trees with specified diameter. *Proc. the 30th Workshop on Graph-Theoretic Concepts in Computer Science, (WG 2004)*, LNCS 3353:33–45, 2004.

[7] S. Nakano and T. Uno. Personal communication. 2004.

[8] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada. Efficient enumeration of all ladder lotteries. In *Proc. The 20th Workshop on Topological Graph Theory, (TGT20)*, 150–151, 2008.

[9] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada. Efficient enumeration of all ladder lotteries. *Technical Report UEC-IS-09-01*, 2009. http://home.hol.is.uec.ac.jp/yamanaka/Tech/UEC-IS-09-01.pdf, submitted to a journal.

On the Number of Crossing-Free Partitions in the Plane

Andreas Razen* Emo Welzl*

Abstract

We show that convex position of a planar set of n points in general position minimizes the number of crossing-free partitions into 1, 2, 3, and $n-3$, $n-2$, $n-1$, n partition classes. A partition of a point set in the plane is called crossing-free, if the convex hulls of the individual parts do not intersect. In this context we also mention a property of point sets in convex position that does not even closely hold in general.

1 Introduction

Let P be a set of n points in the plane. In the following we always assume that P is in *general position*, i.e. no three points are collinear. A partition of P is called *crossing-free* if the convex hulls of the individual parts do not intersect. Note that one may uniquely identify such a crossing-free partition with a plane straight-line embedded graph defined on the vertex set P whose edges are the segments constituting the boundaries of the convex hulls.

We denote by $\mathrm{cfp}(P)$ the number of crossing-free partitions of P, and accordingly write $\mathrm{cfp}_k(P)$ for the number of crossing-free partitions of P into k classes, $1 \le k \le n$. Let Γ_n denote a set of n points in convex position, i.e. Γ_n is the vertex set of a convex n-gon.

It is well-known that for any $n \in \mathbb{N}$

$$\mathrm{cfp}(\Gamma_n) = C_n = \frac{1}{n+1}\binom{2n}{n} \approx 4^n,$$

the n-th Catalan number. In this paper we show that Γ_n minimizes $\mathrm{cfp}_k(P)$ for certain values of k among point sets P with $n = |P|$, while we conjecture that to be true for all k, in particular $\mathrm{cfp}(P) \ge \mathrm{cfp}(\Gamma_n)$. Kreweras [3] calculated

$$\mathrm{cfp}_k(\Gamma_n) = \frac{(n-1)!\; n!}{(k-1)!\; k! \cdot (n-k)!\; (n-k+1)!}\,. \qquad (1)$$

Note that this term is symmetric in the sense that

$$\mathrm{cfp}_k(\Gamma_n) = \mathrm{cfp}_{n-k+1}(\Gamma_n),$$

for all $1 \le k \le n$. We will see that this is not necessarily the case if the points are not in convex position. More precisely we mention a set P of n points with

$$\frac{\mathrm{cfp}_3(P)}{\mathrm{cfp}_{n-2}(P)} = \Theta(n^2).$$

García et al. [2] proved that Γ_n minimizes the number of crossing-free perfect matchings and spanning trees among all sets of n points in general position. Aichholzer et al. [1] extended these results by showing that similar statements also hold for several other graph classes like spanning paths, (pointed) pseudo-triangulations, forests, connected graphs, or all plane graphs. However, it is well-known that triangulations are a prominent counter-example to this pattern. It is open whether Γ_n minimizes the number of crossing-free partitions. Concerning upper and lower bounds it is known [4] that $\mathrm{cfp}(P) = O(12.24^n)$ for any set P, and the *double-chain* allows for $\Omega(5.23^n)$ partitions.

Given a convex polygon S we write ∂S for its boundary and S° for its interior. For a point set P and $Q \subseteq P$ the points from P contained inside the convex hull of Q are denoted by $I^P(Q) := P \cap \mathrm{conv}^\circ(Q)$, and by $E(Q) := Q \cap \partial\mathrm{conv}(Q)$ we refer to the extreme points of Q. For $k \in \mathbb{N}$ we employ the common notion of $\binom{P}{k}$ for the k-element subsets of P. Finally, given a predicate A we denote by $\mathbb{1}_{[A]}$ the indicator function, i.e. $\mathbb{1}_{[A]} = 1$ if A holds and $\mathbb{1}_{[A]} = 0$ otherwise.

2 Partitioning into many classes

As a warm-up observe that the number of crossing-free partitions into $n-1$ and n classes does not depend on the relative position of the points, as long as they are in general position. A partition into $n-1$ classes corresponds to a planar graph with exactly one edge.

Proposition 1 *For a set P of n points in the plane in general position* $\mathrm{cfp}_n(P) = 1$ *and* $\mathrm{cfp}_{n-1}(P) = \binom{n}{2}$.

Let us turn to partitioning P into $n-2$ classes. Before stating a claim about $\mathrm{cfp}_{n-2}(P)$ for arbitrary point sets we will derive that

$$\mathrm{cfp}_{n-2}(\Gamma_n) = 2\binom{n}{4} + \binom{n}{3}.$$

To see this, besides substituting in Identity (1), note that there are only two ways for obtaining a partition of n points into $n-2$ classes, as shown in Figure 1.

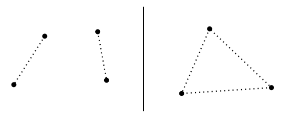


Figure 1: Configurations for $n-2$ partition classes

*Institute of Theoretical Computer Science, ETH Zurich. Authors supported by SNF project number 200021-116741. {razen, emo}@inf.ethz.ch

We have to count the number of pairs of crossing-free matching edges and the (empty) triangles in Γ_n. Every choice of four points allows for two pairs of matching edges, and every set of three points yields a triangle. Hence, the identity from above follows.

In general, however, not every choice of three points in P will result in an empty triangle. Somehow we have to account for the lack of such partitions.

Theorem 2 *Let P be a set of n points in the plane in general position. Then*

$$\mathrm{cfp}_{n-2}(P) = 2\binom{n}{4} + \binom{n}{3} + \sum_{Q\in\binom{P}{3}\,:\,|I^P(Q)|\geq 2} (|I^P(Q)| - 1).$$

In particular we have $\mathrm{cfp}_{n-2}(P) \geq \mathrm{cfp}_{n-2}(\Gamma_n)$.

Proof. Let $R \in \binom{P}{4}$ then either $|E(R)| = 4$ and R has exactly two perfect matchings, or $|E(R)| = 3$ and R has three perfect matchings. Hence, the number of pairs of crossing-free matching edges in P is

$$\sum_{R\in\binom{P}{4}} 2 \cdot \mathbb{1}_{[|E(R)|=4]} + 3 \cdot \mathbb{1}_{[|E(R)|=3]}.$$

We count the sets $R \in \binom{P}{4}$ with $|E(R)| = 3$ by iterating over the extreme points and summing the number of interior points. Then the term above simplifies to

$$2\binom{n}{4} + \sum_{R\in\binom{P}{4}} \mathbb{1}_{[|E(R)|=3]} = 2\binom{n}{4} + \sum_{Q\in\binom{P}{3}} |I^P(Q)|.$$

Finally, adding the number of empty triangles in P yields that $\mathrm{cfp}_{n-2}(P)$ equals

$$\begin{aligned} & 2\binom{n}{4} + \sum_{Q\in\binom{P}{3}} |I^P(Q)| + \sum_{Q\in\binom{P}{3}} \mathbb{1}_{[I^P(Q)=\emptyset]} \\ = \ & 2\binom{n}{4} + \sum_{Q\in\binom{P}{3}} \left(1 + (|I^P(Q)| - 1) \cdot \mathbb{1}_{[|I^P(Q)|\geq 2]}\right) \\ = \ & 2\binom{n}{4} + \binom{n}{3} + \sum_{Q\in\binom{P}{3}} (|I^P(Q)| - 1) \cdot \mathbb{1}_{[|I^P(Q)|\geq 2]}. \end{aligned}$$

Clearly, $(|I^P(Q)|-1)\cdot\mathbb{1}_{[|I^P(Q)|\geq 2|]} \geq 0, \forall\, Q \in \binom{P}{3}$. □

Similarly to this proof we can show that the convex position also minimizes $\mathrm{cfp}_{n-3}(P)$. Note that there are four possibilities to obtain $n-3$ partitions classes, as shown in Figure 2. We have to count the number of empty convex quadrangles, the triangles containing exactly one point, the empty triangles together with a disjoint edge, and finally the number of triples of crossing-free matching edges.

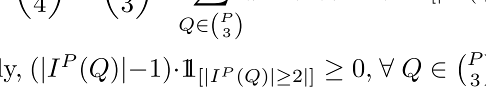


Figure 2: Configurations for $n-3$ partition classes

Clearly, in Γ_n only three of these configurations are possible and hence it is easily seen that

$$\mathrm{cfp}_{n-3}(\Gamma_n) = \binom{n}{4} + 5\binom{n}{5} + 5\binom{n}{6},$$

which may also be verified by evaluating Equation (1) with $k = n - 3$. Convex position uniquely maximizes the number of empty quadrangles (there are exactly $\binom{n}{4}$), while at the same time it minimizes the number of triangles containing a single point and the triples of crossing-free matching edges (any six points in convex position have exactly five perfect matchings).

The detailed analysis is very technical and rather lengthy so we just state the result without proof. The idea for the argument is to consider the 5-element subsets of P and count the corresponding empty triangles together with a disjoint edge. Whenever the triangle contains another point from P we amortize the resulting deficiency by showing that these six points allow for more than five triples of crossing-free matching edges. In order to do this we have to consider all point configurations of six points in the plane of which there are 16. This is also the reason why our arguments so far do not extend to the case of $n-4$ partition classes where we would have to investigate all configurations of eight points (there are several thousands).

Theorem 3 *Let P be a set of n points in the plane in general position. Then* $\mathrm{cfp}_{n-3}(P) \geq \mathrm{cfp}_{n-3}(\Gamma_n)$.

Note that in any set P of n points in general position $\mathrm{cfp}_{n-3}(P) = \Theta(n^6)$. Indeed from Figure 2 we find that $\mathrm{cfp}_{n-3}(P)$ is at most $\binom{n}{6}$ times the maximum number of perfect matchings a set of six points can have (which is 12) plus terms of lower order $O(n^5)$.

3 Partitioning into few classes

We introduce a notion similar to halving edges (or k-edges) in order to identify a crossing-free partition with certain 2-element subsets of P. The benefit is that only a small number of such subsets is needed for describing crossing-free partitions into few classes.

Let A and B be two disjoint convex polygons in the plane. Then there is a line g separating the two sets, i.e. $\forall\, a \in A, b \in B$ the segment with endpoints a and b intersects g. Now rotate g counter-clockwise until it becomes tangent to both polygons simultaneously, see Figure 3.

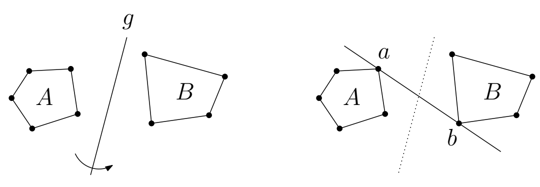


Figure 3: Construction of a separating segment

Suppose that no three extreme points of the polygons are collinear then we obtain a unique (extreme) point $a \in A$ and a unique (extreme) point $b \in B$. The segment given by these endpoints a and b is the *separating segment* of the polygons A and B.

Proposition 4 *For P a set of n points in the plane in general position* $\mathrm{cfp}_1(P) = 1$ *and* $\mathrm{cfp}_2(P) = \binom{n}{2}$.

Proof. To partition P into two crossing-free parts pick a segment defined by two points $p, q \in P$. Then slightly rotate the line through pq clockwise around a point between p and q in order to obtain a partition.

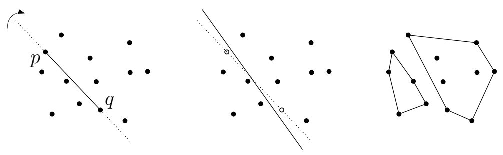


Figure 4: Crossing-free partition into 2 classes

Observe that the separating segment for two given polygons is unique because of general position. Thus, for any point set P we have $\mathrm{cfp}_2(P) = \binom{n}{2}$, regardless of the relative position of the points. □

When partitioning P into more parts it becomes quite handy to use colors for the classes. Consider the set of surjective maps $\chi : P \to \{1, \ldots, k\}$. Two such k-colorings χ_1, χ_2 are equivalent if there is a permutation π of the colors such that $\chi_1 = \pi \circ \chi_2$. There is a bijection between the set of equivalence classes of these colorings and the partitions of P into k parts.

If the partition induced by a coloring is crossing-free then for each pair of distinct colors i and j there is a separating segment associated with it. We will indicate the halfspace containing the points of color i (and j, respectively) by drawing i (and j) next to the segment's endpoint in the corresponding halfspace.

In case of $k = 3$ partition classes any two separating segments share a color, in particular they cannot cross in an interior point as can be seen from Figure 5.

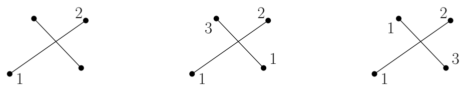


Figure 5: Separating segments cannot cross

Consider two disjoint segments that w.l.o.g. both contain color 1. Their convex hull is either a quadrangle or a triangle, and the only possibilities for the corresponding colorings are shown in Figure 6.

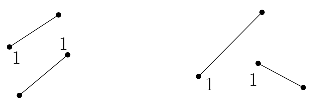


Figure 6: Disjoint segments with a common color

There are two ways to partition Γ_n into three parts. Either every class can be separated from both other classes simultaneously by exactly one line or there is a class that needs two lines; see Figure 7 (partition classes are drawn as dotted lines whereas the separating segments are solid).

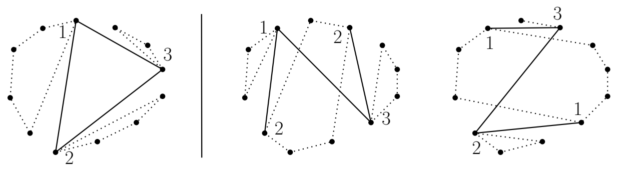


Figure 7: Partitions and separating segments

The crucial observation here is that we only need the endpoints of the separating segments and their color in order to reconstruct the underlying partition. Clearly, every three points in Γ_n yield a crossing-free partition into three classes where the corresponding separating segments build an (empty) triangle. Furthermore, every choice of four points in Γ_n allows for two different ways of obtaining a partition into three crossing-free parts. This shows $\mathrm{cfp}_3(\Gamma_n) = 2\binom{n}{4} + \binom{n}{3}$.

For general P we will have to account for non-empty triangles. A set $Q \in \binom{P}{3}$ with empty interior certainly contributes 1 to $\mathrm{cfp}_3(P)$, see Figure 8. Note that there is only one way to assign the colors, then the regions for the partition classes are uniquely determined.

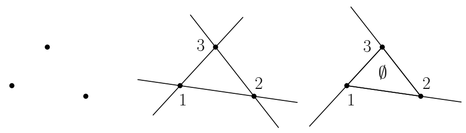


Figure 8: Construction from an empty triangle

Now consider four points in convex position. There are two ways to obtain a pair of disjoint segments from such points which we call *parallel segments* (even if the lines intersect). The color of the points is determined as described above. Figure 9 shows how to obtain a partition into three classes from such segments.

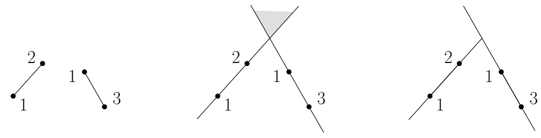


Figure 9: Construction from parallel segments

Finally, we consider configurations not appearing in Γ_n which will compensate for triangles with points in their interior and pairs of segments whose convex hulls is a triangle. The latter we call *spearing segments.*

There are three ways to obtain a pair of disjoint segments from four points with triangular convex hull. Fix a pair then there is exactly one coloring of their endpoints, see Figure 10. It remains to to determine the partition of the plane which the segments induce.

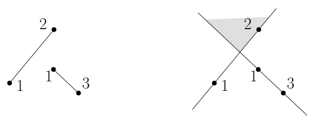


Figure 10: Construction from spearing segments

It turns out that in general we cannot color the gray region with a single color so that distinct pairs of spearing segments induce distinct partitions. We need to use the third segment which will determine the mixed coloring of the gray region as seen in Figure 11.

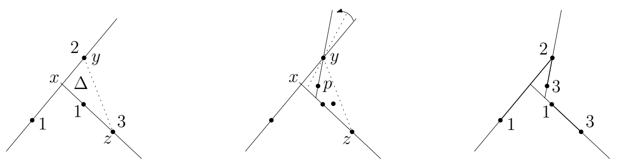


Figure 11: Constructing the third separating segment

For this purpose consider the segments containing color 1 and let y, z be the endpoints with color 2 and 3. Let x be the intersection of the lines through these segments and Δ the triangle given by x, y and z. Now rotate the line through x and y counter-clockwise around y. Let $p \in P \cap (\Delta^\circ \cup \{z\})$ be the first point of P that gets hit during this rotation. We color p with 3 and define the third separating segment by its endpoints y and p. This results in a partition of the plane into four convex regions where the unbounded areas each contain exactly one crossing-free partition class and the bounded triangle is empty by construction.

Theorem 5 *Let P be a set of n points in the plane in general position. Then* $\mathrm{cfp}_3(P) \geq \mathrm{cfp}_3(\Gamma_n)$.

Proof. An empty triangle contributes 1 to $\mathrm{cfp}_3(P)$, four points either two pairs of parallel or three pairs of spearing segments. Note that the induced partitions are pairwise distinct. Hence, $\mathrm{cfp}_3(P)$ is at least

$$\sum_{Q\in\binom{P}{3}\,:\,I^P(Q)=\emptyset} 1 \quad + \sum_{Q\in\binom{P}{4}\,:\,|E(Q)|=4} 2 \quad + \sum_{Q\in\binom{P}{4}\,:\,|E(Q)|=3} 3$$

$$= 2\binom{n}{4} + \binom{n}{3} + \sum_{Q\in\binom{P}{3}} (|I^P(Q)|-1)\cdot \mathbb{1}_{[|I^P(Q)|\geq 2]}.$$

The identity follows as in the proof of Theorem 2. The last expression is exactly $\mathrm{cfp}_{n-2}(P)$, therefore we get $\mathrm{cfp}_3(P) \geq \mathrm{cfp}_{n-2}(P) \geq \mathrm{cfp}_{n-2}(\Gamma_n) = \mathrm{cfp}_3(\Gamma_n)$. □

Many partitions into three classes

Recall that $\mathrm{cfp}_k(\Gamma_n) = \mathrm{cfp}_{n-k+1}(\Gamma_n)$ and Theorem 2 implies that $\mathrm{cfp}_{n-2}(P) = \Theta(n^4)$, since $|I^P(Q)| \leq n$. We consider a point set that is also known for having a large number of halving edges. For its construction start with the following set $P(1)$ of six points, a triangle with a smaller similar triangle inscribed and slightly perturbed, see Figure 12.

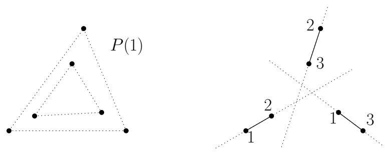


Figure 12: Three parallel separating segments

Note that $P(1)$ has a triple of separating segments we did not account for in Theorem 5. If a point set P contains $P(1)$ (or an affine copy) as a subset, then these three parallel segments additionally contribute to $\mathrm{cfp}_3(P)$ if and only if the bounded region that the lines define is empty. In this case the triple is *valid.*

Assuming a proper underlying coordinate system, we call a point set *ε-flattened copy of P* if it is obtained by multiplying the y-coordinates of all points in P by ε. The construction starts with $P(1)$ and for $\ell \geq 1$ recursively builds $P(\ell+1)$ by arranging three copies of ε_ℓ-flattened point sets of $P(\ell)$ (denoted by $P^i_{\varepsilon_\ell}(\ell)$, for $i = 1, 2, 3$) in a tripod, as shown in Figure 13.

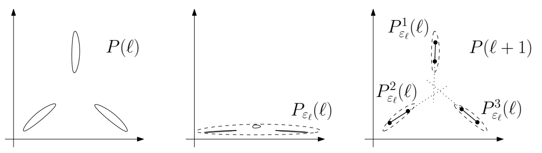


Figure 13: Recursive construction of $P(\ell)$

By choosing $\varepsilon_\ell > 0$ small enough we can guarantee that for any choice of six points, two from each $P^i_{\varepsilon_\ell}(\ell)$ with $i = 1, 2, 3$, we obtain a valid triple of parallel segments. Indeed we can always avoid two points having the same x-coordinate by applying a small perturbation, and therefore the slope of any line through two points can be made arbitrarily small by flattening. For estimating $\mathrm{cfp}_3(P(\ell))$ write $|P(\ell)| = 2\cdot 3^\ell = n(\ell)$ and count the valid triples of parallel segments that can be obtained from $P(\ell)$. This results in

$$\mathrm{cfp}_3(P(\ell)) \geq \sum_{k=1}^{\ell} 3^{k-1}\cdot\binom{\frac{n(\ell)}{3^k}}{2}^3 = \frac{n(\ell)^6}{5808} + O(n(\ell)^5),$$

which is up to a constant the maximum attainable.

References

[1] O. Aichholzer, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber, On the number of plane graphs, *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms* (2006), 504–513.

[2] A. García, M. Noy, and J. Tejel, Lower bounds on the number of crossing-free subgraphs of $\mathrm{K_N}$, *Comput. Geom. Theory Appl.* **16**(4), (2000), 211–221.

[3] G. Kreweras, Sur les partitions non croisées d'un cycle, *Discrete Math.* **1**(4), (1972), 333–350.

[4] M. Sharir and E. Welzl, On the Number of Crossing-Free Matchings, Cycles, and Partitions, *SIAM J. Comput.* **36**(3), (2006), 695–720.

Counting the Number of Embeddings of Minimally Rigid Graphs

Ioannis Z. Emiris[*] Antonios Varvitsiotis[†]

Abstract

We exploit the Hennenberg constructions of Laman graphs and of the 1-skeleta of simplicial polyhedra in order to study their respective number of Euclidean embeddings. This leads to the general upper bounds of $2^k 4^{n-k-4}$ for planar and $2^{k+1} 8^{n-k-5}$ for spatial graphs, where k is the number of degree-2 and degree-3 vertices, respectively, and n denotes the number of vertices. These improve the known bounds only when k is large. For the planar case we also give a new general lower bound of about $(2.37)^n$, which improves upon the existing ones. Our approach yields tight bounds for all cases up to $n = 8$ in the plane, and up to $n = 6$ in space. As a corollary, we present a simple proof of the Hirsch conjecture for simplicial 3-polytopes.

1 Introduction

In this paper we deal with the computation of the number of distinct planar and spatial Euclidean embeddings of minimally rigid graphs up to rigid motions.

For the planar and spatial case, the best known general upper bounds are $\binom{2n-4}{n-2} \approx 4^{n-2}/\sqrt{\pi(n-2)}$ and $\frac{2^{n-3}}{n-2}\binom{2n-6}{n-3} \approx 8^{n-3}/\big((n-2)\sqrt{\pi(n-3)}\big)$ respectively, where n denotes the number of vertices. These bounds were obtained in [2] by distance matrix theory, complex algebraic geometry, and determinantal varieties. The same paper gave some lower bounds, the strongest being obtained by the Desargues fan construction, namely $2 \cdot 12^{n/3-1} \simeq 2.29^n/6$. [1]

The approach of [8] was to define a square polynomial system, obtained by the edge length constraints, whose real solutions correspond precisely to the different embeddings (see systems f, f' in section 3). Mixed volume bounds the number of complex roots of an arbitrary polynomial system; by exploiting the sparseness of the equations, it often yields tighter bounds than the classic Bézout bound. By suitably writing the equations expressing rigid graphs, a structure becomes manifest, thus leading to the bound of 4^{n-2}.

Our approach is mainly constructive and uses bounds on the number of embeddings induced by each Hennenberg step. In the next section, we provide an alternative proof than the one in [8], for the exact number of embeddings when $n = 6$. We establish tight bounds for $n = 7, 8$ equal to 64 and 128, respectively. By extending the caterpillar of [2] to $K_{3,3}$, we obtain a better lower bound of about $(2.37)^n$, for $n \geq 10$. Lastly, we establish an upper bound of $2^k 4^{n-k-4}$, where k is the number of vertices of degree 2. This becomes 4^{n-4} as a function of n.

We also study the number of spatial embeddings of the 1-skeleton (or edge graph) of (convex) simplicial polyhedra. These polyhedral graphs are known to be generically minimally rigid in $\mathbb{R}^3$ [4]. In Sec. 3, we produce all such graphs by generalized Hennenberg steps. We obtain tight bounds on the number of spatial embeddings for graphs of 4, 5 and 6 vertices respectively, and we present a general upper bound of $2 \cdot 8^{n-5}$ for graphs of n vertices. Our bound is much easier to demonstrate by relying on simple facts concerning the construction of these graphs. We also obtain a general bound of $2^{k+1} 8^{n-k-5}$, where k denotes the number of vertices of degree 3. Lastly, we offer a simple proof of the Hirsch conjecture in $d = 3$, for simplicial polytopes, in Sec. 4.

2 Planar embeddings of Laman graphs

Every Laman graph, or minimally rigid planar graph, is characterized by having $2n - 3$ edges and all of its subgraphs on $k < n$ vertices having $\leq 2k - 3$ edges. Equivalently a graph is Laman iff it has a Hennenberg construction starting from a triangle followed by a succession of Hennenberg-1 (or H_1) and Hennenberg-2 (or H_2) steps. Since two circles intersect generically in 2 points, a H_1 step at most doubles the number of embeddings. A H_2 step adds 2 new quadratic equations to the corresponding polynomial system and thus at most quadruples the number of embeddings.

We write $\triangle s_1 \ldots, s_n,\ s_i \in \{1, 2\}$ for the class of Laman graphs whose Hennenberg step sequence is $s_1, \ldots, s_n$. A graph is of type H_1 if it can be constructed using only H_1 steps, but it is of type H_2 if its construction includes at least one H_2 step. Now,

[*]Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece. The author is supported through PENED 2003 program, contract nr. 70/03/8473. The program is co-funded by the EU – European Social Fund (75% of public funding), national resources – General Secretariat of Research and Technology of Greece (25% of public funding) as well as the private sector, in the framework of Measure 8.3 of the Community Support Framework.

[†]MPLA, Graduate Program in Logic, Algorithms and Computation, National and Kapodistrian University of Athens, Greece.

[1]This corrects the exponent of the original statement.

it is not difficult to verify that:

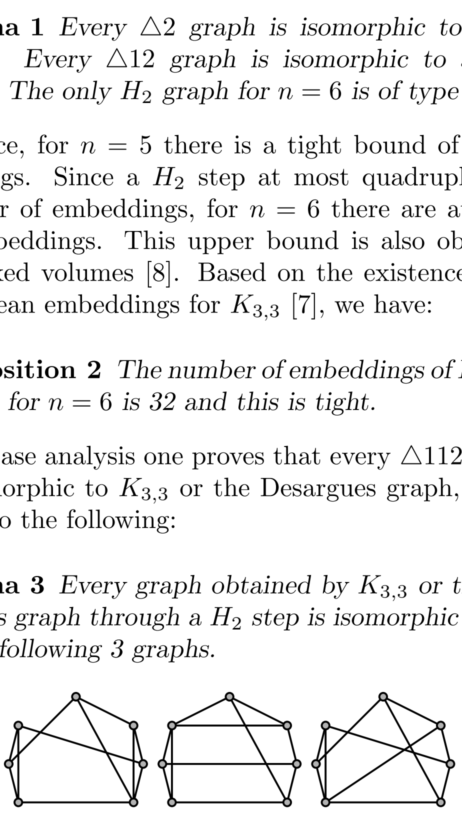
Lemma 1 *Every $\triangle 2$ graph is isomorphic to a $\triangle 1$ graph. Every $\triangle 12$ graph is isomorphic to a $\triangle 11$ graph. The only H_2 graph for $n = 6$ is of type $\triangle 112$.*

Hence, for $n = 5$ there is a tight bound of 8 embeddings. Since a H_2 step at most quadruples the number of embeddings, for $n = 6$ there are at most 32 embeddings. This upper bound is also obtained by mixed volumes [8]. Based on the existence of 32 Euclidean embeddings for $K_{3,3}$ [7], we have:

Proposition 2 *The number of embeddings of Laman graphs for $n = 6$ is 32 and this is tight.*

By case analysis one proves that every $\triangle 112$ graph is isomorphic to $K_{3,3}$ or the Desargues graph, which leads to the following:

Lemma 3 *Every graph obtained by $K_{3,3}$ or the Desargues graph through a H_2 step is isomorphic to one of the following 3 graphs.*



Theorem 4 *The number of embeddings of Laman graphs on 7 vertices is 64 and this bound is tight.*

Proof. Let us first establish the upper bound. Any Laman graph G on 7 vertices corresponds to $G_6 s_i$, where G_6 is a Laman graph on 6 vertices and $s_i \in \{1, 2\}$. Now, if $s_i = 1$ then, by Proposition 2, we are done. If $s_i = 2$, there are two cases for G_6: If G_6 is of type H_1, then G has at most $2^4 \cdot 4 = 64$ embeddings. Otherwise, G_6 is isomorphic to $K_{3,3}$ or the Desargues graph and we refer to Lemma 3; for all graphs of Lemma 3, the mixed volume of the corresponding polynomial system was computed using the PHCpack[2] and was found to be 64.

Since $K_{3,3}$ may actually have 32 embeddings, and since we can choose edge lengths such that a H_1 step exactly doubles the number of embeddings, we also obtain a lower bound of 64. □

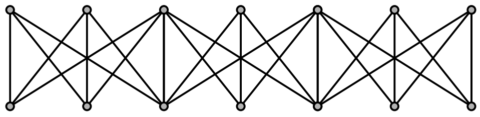
Theorem 5 *The number of embeddings of Laman graphs for $n = 8$ is 128 and this bound is tight.*

Proof. We have a lower bound of 128 from Theorem 4. E. Tsigaridas has implemented in SAGE[3], using the module for graph theoretic operations, a routine which constructs all Laman graphs for up to $n = 8$, where isomorphic graphs are identified at every stage. We keep one representative, whose Hennenberg sequence contains the least number of H_2 steps. We observe that all graphs for $n = 8$ are either of H_1 type or of type $\triangle 11112$ [6], hence the upper bound of $2^5 \cdot 4 = 128$. □

We pass to general bounds. We establish a new lower bound by constructing a $K_{3,3}$ caterpillar, thus extending the Desargues caterpillar [2].

Theorem 6 *There exist edge lengths for which the $K_{3,3}$ caterpillar construction has $32^{\frac{n}{4}-\frac{1}{2}} \approx (2.37)^n$ embeddings, for $n \geq 10$.*

Proof. We glue copies of $K_{3,3}$. Each new copy has exactly one edge in common with the previous one; see an example with 3 copies:



The resulting graph has the Laman property and, since each $K_{3,3}$ copy adds 4 vertices and has 32 embeddings, the statement follows. □

We summarize the known results and their implications up to $n = 10$. The Desargues fan [2] yields the lower bound for $n = 9$, and Theorem 6 yields the lower bound for $n = 10$.

n	5	6	7	8	9	10
	8	32	64	128	$[288, 512]$	$[1024, 2048]$

Theorem 7 *Let G be a Laman graph with $n \geq 7$ vertices, and k denote the number of vertices of degree 2. Then, the number of planar embeddings of G is bounded above by $2^k 4^{n-k-4}$.*

Proof. Let G_7 denote a Laman graph on 7 vertices with 64 embeddings. We construct a Hennenberg sequence for G as follows: if there exists a vertex of degree 2, perform a H_1 step in reverse i.e. remove the vertex and its two edges. This does not affect any of the existing degree-2 vertices (because the subgraph is also Laman), although it may create new ones. In the worst case, the removal of k degree-2 vertices results to every vertex having degree ≥ 3. It is easy to show that there exists a vertex v of degree 3; let x, y, z be its neighbours. These cannot form a triangle because the Laman condition would be violated for the subgraph of v, x, y, z. So we perform a H_2 step in reverse i.e. remove v and add an edge between two of its neighbours which are not connected. The removal of v may create one new vertex of degree 2 but, in the worst case, no such vertex is created, so G has the following Hennenberg sequence:

$$G_7, s_8, \ldots, s_{n-k-7}, \underbrace{11\ldots11}_{k}, \quad s_i \in \{1, 2\}.$$

[2]http://www.math.uic.edu/~jan/
[3]`http://www.sagemath.org/`

Thus, the number of embeddings is $\leq 2^k 4^{n-k-4}$. □

Corollary 8 *The number of planar embeddings of a Laman graph on $n \geq 7$ vertices is bounded by 4^{n-4}.*

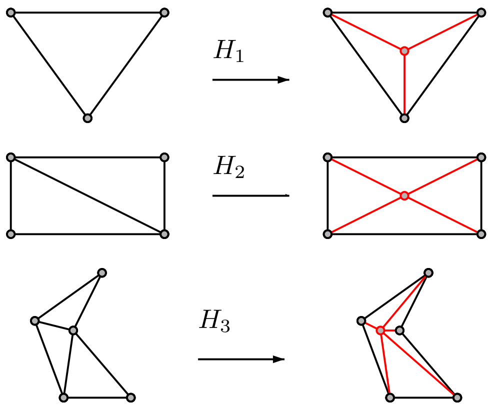
3 1-skeleta of simplicial polyhedra

This section extends the planar Hennenberg steps for constructing all 1-skeleta of simplicial polyhedra, thus bounding the number of embeddings of such graphs.

Since 3 spheres intersect generically in two points, a H_1 step at most doubles the number of spatial embeddings. The latter can also be proven by mixed volumes, extending the proof in [8] for the planar H_1. Since a H_2 and a H_3 step add 3 new quadratic equations to the polynomial system obtained by edge lengths, its Bézout bound is multiplied by 8, hence each such step, at worst, multiplies the number of embeddings by 8.

Recall that the 1-skeleton of a (convex) polyhedron P is minimally rigid in $\mathbb{R}^3$ iff P is simplicial, e.g. [4].

Consider any $k+2$ vertices forming a cycle, which contains at least $k-1$ diagonals. The extended Hennenberg-k (H_k) step, for $k = 1, 2, 3$, corresponds to adding a vertex, connecting it to the $k+2$ vertices, and removing $k-1$ diagonals among them:

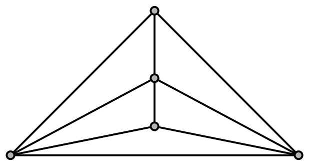


Proposition 9 [3] *Graph G is the 1-skeleton of a simplicial polyhedron in $\mathbb{R}^3$ iff it has a construction that begins with the 1-skeleton of the 3-simplex followed by any sequence of H_1, H_2, H_3 steps.*

In $\mathbb{R}^3$, to discard translations and rotations, we fix a facet of the polytope. The only simplicial polytope on 4 vertices is the 3-simplex and its 1-skeleton has exactly two embeddings.

Theorem 10 *The 1-skeleton of a simplicial polyhedron with $n = 5$ has at most 4 embeddings and this bound is tight.*

Proof. It is known (e.g. [3]) that all 1-skeleta of simplicial polyhedra with $n = 5$ are isomorphic to the following:



This graph is obtained by the 1-skeleton of the 3-simplex through a H_1 step. □

In order to establish an upper bound on the number of embeddings we compute the mixed volume of the polynomial system $f(x)$ obtained by the edge length constraints. To discard translations and rotations, we fix the facet of vertices v_1, v_2, v_3 and obtain the system:

$$f(x) = \begin{cases} x_i = a_i, & i = 1,2,3 \\ y_i = b_i, & i = 1,2,3 \\ z_i = c_i, & i = 1,2,3 \\ (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 - l_{ij}^2 = 0, \\ \forall (i,j) \in E - \{(v_1,v_2),(v_2,v_3),(v_3,v_1)\} \end{cases}$$

Let $v = (0,0,0,0,0,0,0,0,0,-1,\ldots,-1)$, the corresponding face system is:

$$\partial_v f(x) = \begin{cases} x_i = a_i, & i = 1,2,3 \\ y_i = b_i, & i = 1,2,3 \\ z_i = c_i, & i = 1,2,3 \\ (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 = 0, \\ \forall (v_i,v_j) \in E \text{ with } i,j \neq 1,2,3 \\ x_i^2 + y_i^2 + z_i^2 = 0, \\ \forall (v_i,v_k) \in E \text{ with } k = 1,2,3 \end{cases}$$

This system has

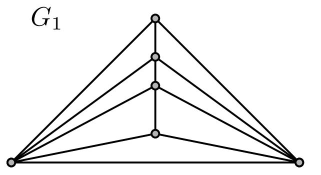
$$(\underbrace{a_1,b_1,c_1}_{v_1},\ldots,\underbrace{a_3,b_3,c_3}_{v_3},\underbrace{1,1,i\sqrt{2}}_{v_4},1,1,i\sqrt{2},\ldots,\underbrace{1,1,i\sqrt{2}}_{v_{|V|}})$$

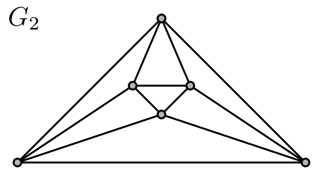
as a solution in $(\mathbb{C}^*)^n$. Hence, by the Second Theorem in [1], the mixed volume is not a tight bound on the number of solutions in $(\mathbb{C}^*)^n$. This was also observed about the corresponding system in the planar case [8]. To remove spurious solutions (at toric infinity), for each $i = 1,\ldots,|V|$, we introduce variable $s_i = x_i^2 + y_i^2 + z_i^2$, just as in the planar case. This yields the equivalent system:

$$f'(x) = \begin{cases} x_i = a_i, & i = 1,2,3 \\ y_i = b_i, & i = 1,2,3 \\ z_i = c_i, & i = 1,2,3 \\ s_i - x_i^2 - y_i^2 - z_i^2 = 0, & i = 1,\ldots,|V| \\ s_i + s_j - 2x_ix_j - 2y_iy_j - 2z_iz_j - l_{ij}^2 = 0, \\ \forall (i,j) \in E - \{(v_1,v_2),(v_2,v_3),(v_3,v_1)\} \end{cases}$$

Theorem 11 *The 1-skeleton of a simplicial polyhedron on 6 vertices has at most 16 embeddings and this bound is tight.*

Proof. There are only two non-isomorphic polyhedral graphs G_1, G_2 (e.g. [3]) corresponding to a 1-skeleton of a simplicial polyhedron for $n = 6$:





Since all facets of G_2 are symmetric, we fix one and compute a mixed volume of 16 for $f'(x)$. Fixing one of the facets of G_1 we compute a mixed volume of 8. So the upper bound is 16.

Now G_2 is the graph of the Cyclohexane molecule, which has 16 different real spatial embeddings, e.g. [5]. □

Theorem 12 *Let G be the 1-skeleton of a simplicial polyhedron with $n \geq 6$ vertices and let k denote the number of vertices of degree 3. Then the number of embeddings of G is bounded by $2^{k+1}8^{n-k-5}$.*

Proof. Let G_6 denote a graph with 6 vertices and 16 embeddings. To construct a Hennenberg sequence for G we first remove all existing degree-3 vertices by performing k H_1 steps in reverse. In the worst case, the new graph contains no degree-3 vertex, and it can be shown that it contains one of degree 4 or 5 [3]. Hence, in the worst case, G has the following Hennenberg sequence:

$$G_6, s_7, \ldots, s_{n-k-6}, \underbrace{11 \ldots 11}_{k}, \quad s_i \in \{2, 3\}.$$

Thus the number of embeddings is $\leq 2^{k+1}8^{n-k-5}$. □

Corollary 13 *The number of spatial embeddings of any 1-skeleton of a simplicial polyhedron on n vertices where $n \geq 6$ is bounded above by $2 \cdot 8^{n-5}$.*

4 The Hirsch conjecture

The diameter of a graph, denoted by $\delta(G)$, is the longest path between any two vertices of the graph. Let $d(u, v)$ be the distance between vertices u, v and let $\Delta(d, f)$ denote the maximal diameter of the 1-skeletons of d-dimensional polyhedra P with f facets. In 1957, Hirsch conjectured that $\Delta(d, f) \leq f - d$; this is known to be true for $d < 4$ and for various special cases but the general status of the problem is open.

The 1-skeleta of simplicial polyhedra correspond to triangulations and, since each Hennenberg step increases the number of triangles by 2, we obtain:

Lemma 14 *A simplicial polyhedron has an even number of facets.*

Theorem 15 *The Hirsch conjecture is true for the 1-skeleta of simplicial polyhedra.*

Proof. By induction on the number $2k$ of facets of a simplicial polyhedron. For $k = 2$, $\Delta(3, 4) \leq 1$. Let P be a polyhedron with $2k + 2$ facets and let $S, P_1, \ldots, P_{k-2}, P_{k-1} = P$ be its Hennenberg sequence, where S denotes the 1-skeleton of the tetrahedron. By hypothesis $\delta(P_{k-2}) \leq 2k - 3$. Let n be the new vertex, u, v old vertices, and a a neighbour of n. There are 3 cases:
(i) P_{k-1} is obtained by P_{k-2} through an H_1 step. A H_1 step removes no edges so $d(u, v) \leq 2k - 3$ in P_{k-1}, thus $d(u, n) = d(u, a) + 1 < 2n - 1$.
(ii) P_{k-1} is obtained by P_{k-2} through a H_2 step and let ac be the removed edge. By hypothesis $d(u, v) \leq 2k - 3$ in P_{k-2}. If $(ac) \notin P$ then the claim follows; if $(ac) \in P$ then, for $P' = (P - (ac)) \cup (c, n) \cup (n, a)$, we have $length(P') = length(P) + 1 \leq 2k - 2$. Finally, $d(u, n) = d(u, a) + 1 \leq 2k - 1$.
(iii) P_{k-1} is obtained by P_{k-2} through a H_3 step. Let $(a, c), (b, d)$ be the edges removed, and P a path between u, v in P_{k-2} with $length(P) \leq 2k - 3$. There are 2 case two consider: Edges $(a, c), (b, d)$ either share a common vertex or not. In both of these cases we can show that for any pair of old vertices u, v we have $d(u, v) \leq 2k - 2$ in P_{k-1}, so again $d(u, n) = d(u, a) + 1 \leq 2k - 1$.

□

References

[1] D.N. Bernstein. The number of roots of a system of equations, *Funkc. Anal.i Prilozen*, 9(3), 1975.

[2] C. Borcea and I. Streinu. On the number of embeddings of minimally rigid graphs, *Discrete Computat. Geom.*, 31(2), 287–303, 2004.

[3] R. Bowen and S. Fisk. Generation of triangulations of the sphere, *Math. Comput.*, 21(98), 250–252, 1967.

[4] H. Gluck. Almost all simply connected closed surfaces are rigid, *Geometric Topology*, Lecture Notes in Math., Volume 438, 225–239, Springer, Berlin, 1975.

[5] I.Z. Emiris and B. Mourrain. Computer algebra methods for studying and computing molecular conformations, *Algorithmica*, 372–402, 1999.

[6] E. Tsigaridas. Personal communication, 2009.

[7] D. Walter and M.L. Husty. On a nine-bar mechanism, its possible configurations and conditions for flexibility, In J.-P. Merlet and M. Dahan, eds., *Proc. IFFToMM*, Besançon, France, 2007.

[8] R. Steffens and T. Theobald. Mixed volume techniques for embeddings of Laman graphs, *Euro-CG 2008, Collection of Abstracts*, 25–28, Nancy, France.

Reconstructing Points on a Circle from Labeled Distances

David Bremner* Erik D. Demaine† Perouz Taslakian‡ Godfried Toussaint‡

Abstract

In the *labeled beltway problem*, we are given a set of points with unknown coordinates, their clockwise ordering around the circumference of a circle, and a coloring of the edges between pairs of points. The goal is to embed the points around a unit circle while satisfying the constraint that two incident edges have the same geodesic length on the circle if and only if they have the same color. We give a polynomial-time algorithm to find such an embedding or to determine that no such embedding exists.

1 Introduction

In this paper we study a reconstruction problem where we are required to embed a set of points along the circumference of a circle that satisfy some constraints on the pairwise interpoint intervals.

Reconstruction problems have connections with many areas such as music theory, crystallography, and DNA sequencing [3, 8]. One variation, called the *turnpike problem*, dates back to the 1930's in the area of X-ray crystallography, where the objective was to reconstruct the coordinates of the atoms in a crystal. More recently, reconstruction problems have become important in the field of DNA *sequencing* — determining the pattern of the amino acids that constitute a strand of DNA. Given a DNA molecule, exposing it to a special kind of enzyme (called *restriction enzyme*) divides it into pieces of different lengths. The lengths of these fragments can be measured with standard techniques, and the challenge is to reconstruct the original ordering of these fragments in the DNA molecule. In molecular biology, this reconstruction problem is called the *partial digest* problem and is equivalent to the turnpike problem in crystallography.

In the context of music, the geodesic distances (along the circumference) between the points on a circle correspond to duration (time) intervals between onsets, in the case of rhythm, and pitch intervals for chords, scales, and melodies [9]. Questions concerning the existence and constructibility of melodies and rhythms from partial information is a well-studied problem in music theory, music perception, and music information retrieval. One popular method of encoding rhythms and melodies with only partial information is via rhythmic and melodic *contours.* The rhythmic contour is defined as the pattern of successive relative changes of durations in a rhythm. A pitch (or melodic contour) is defined in the same way [7]. Two types of contours have received a great deal of attention in the music literature: the *adjacent* contours that use the matrix of distances between pairs of consecutive points on the circle, and the *full* contours that use the entire matrix of distances between all pairs of points on the circle. An interesting composition problem in music theory is whether a given set of intervals (adjacent or full) admits a realization as a melody or rhythm [6]. Demaine et al. [1] give efficient algorithms for determining whether a rhythm may be reconstructed. In [2], Demaine et al. characterize all rhythms where each distance defined by pairs of points appears a unique number of times. Their characterization also provides an algorithm for generating such pointsets. In this paper we are concerned with a further reduction of information compared to that contained in the contours explored in the music literature to date. Rather than specifying whether interval lengths increase, decrease, or stay the same, we only use one bit of information for comparing two intervals: either they have the same duration or they do not. In what follows, we describe a reconstruction problem we call the *labeled beltway problem*, and provide a polynomial-time solution.

From a theoretical perspective, problems related to reconstructing sets from interpoint distances are, in general, computationally challenging. Even when the points are restricted to a line (turnpike or partial digest problem), the complexity of the problem remains unknown. One of the main difficulties of these problems lies in the fact that while a given pointset uniquely defines a multiset of distances, the inverse is not always true: there may be many pointsets defining a given multiset of distances; such pointsets are known as *homometric* sets. Lemke et al. [3] study the computational and combinatorial complexity of reconstruction problems. For a given set of $\binom{n}{2}$ distances, they give upper and lower bounds on the number of mutually noncongruent and homometric n-point sets that realize these distances in $\mathbb{R}^d$. They also show that the decision problem of whether a multiset of $\binom{n}{2}$ distances is realized by n points in $\mathbb{R}^d$ (for arbitrary dimensions d) is NP-complete.

Lemke et al. [3] call the version of the reconstruction

*University of New Brunswick, bremner@unb.ca
†Massachusetts Institute of Technology, edemaine@mit.edu
‡McGill University, {perouz,godfried}@cs.mcgill.ca

problem where the points lie on a circle the *beltway problem*. Here we consider a variation of the beltway problem, the *labeled beltway*, where the edges defined by the points along the circle are assigned labels. The problem can be stated as follows: given a set of points and their clockwise order around a circle, we want to find an embedding of the points on the circumference of a unit circle subject to some constraints. The constraints involve the *geodesic distance* between pairs of points, that is, the length of the shortest path between two points along the circumference of the circle. The distance constraints are given by a labeling of the edges defined by pairs of points such that, if two distinct edges sharing a common endpoint have the same label, then their geodesic distances must be equal. Moreover, if two distinct edges sharing a common endpoint have distinct labels, then their geodesic distances must be distinct. We give a polynomial-time solution to the labeled beltway problem by reducing it to solving a set of linear equations and nonequations.

Note that if, together with the ordering, we were given all the distances between pairs of points, the problem becomes easy in the plane, as well as on the circle. If the points are embedded on a circle, then we have two cases: if the lengths of the convex-hull edges sum to 1, then the given edge lengths measure the clockwise distance between pairs of points consecutive around the circumference, and we can embed. Otherwise, the longest edge of the convex hull equals the sum of the lengths of the remaining edges; in this case, invert the length of the longest edge around 1; all the edges now have clockwise distances and we can again embed.

2 Definition and Notation

Let $\{p_0, p_1, \ldots, p_{n-1}\}$ be a set of points embedded on a unit circumference circle such that $p_{(i+1) \bmod n}$ is the closest point to p_i in the clockwise traversal along the circumference starting from p_i for $i = 0, 1, \ldots, n-1$.[1] The *clockwise distance* between p_i and p_j, denoted by $\overrightarrow{d}(p_i, p_j)$, is the length of the clockwise traversal along the circumference of the circle from p_i to p_j. The *geodesic distance* between p_i and p_j, denoted by $d(p_i, p_j)$, is the length of the minimum clockwise and counterclockwise traversals along the circumference of the circle between p_i and p_j. We say that an edge p_ip_j is *oriented clockwise* if its geodesic distance is the length of the traversal from p_i to p_j along the circumference of the circle; it is *oriented counterclockwise* if its geodesic distance is the length of the traversal from p_j to p_i along the circumference. If the length of an edge is $\frac{1}{2}$, then the edge can be oriented either way; but otherwise, its orientation is forced. We specify the constraints on the geodesic distances between pairs of points by associating these distances with edges of the complete graph on the n points.

[1]Henceforth, all index arithmetic is modulo n.

3 Reduction to Linear Equations and Nonequations

We first state the constraints on the edges of the labeled complete graph more formally as follows: for every three distinct points p_i, p_j, p_k on the circle, if the edge p_ip_j has the same label as p_jp_k, then the geodesic distance between p_i and p_j is equal to the distance between p_j and p_k, that is, $d(p_i, p_j) = d(p_j, p_k)$ (*isometry constraints*); moreover, two edges with different labels must have different geodesic lengths (*anisometry constraints*).

We show how to reconstruct the points given isometry and anisometry constraints between pairs of edges that do not cross (that is, edges whose endpoints appear together in the cyclic order). In particular, such constraints capture all constraints, solving the problem. Our solution is based on representing the constraints by linear equations ($\sum_i a_ix_i = b_1$), linear inequalities ($\sum_i a_ix_i \leq b_i$ and $\sum_i a_ix_i < b_i$), and linear nonequations ($\sum_i a_ix_i \neq b_i$) on n variables. We then solve this system by reducing to a sequence of linear programs with only linear equations and inequalities.

We parameterize the desired embedding of the n points on the circle by variables $x_0, x_1, \ldots, x_{n-1}$, where $x_i = \overrightarrow{d}(p_i, p_{i+1})$, the clockwise distance from p_i to p_{i+1}. These n variables determine an embedding up to rotation provided that they satisfy the following constraints (defining an open $(n-1)$-simplex):

$$\begin{aligned} \sum_{i=0}^{n} x_i &= 1, \\ x_i &> 0 \qquad \text{for } i = 0, 1, \ldots, n-1. \end{aligned}$$

The positivity constraints force the points to embed to distinct locations in the correct cyclic order.

The challenge in representing an isometry or anisometry constraint among geodesic distances is that a geodesic distance between two points p_i and p_j may be realized by either the clockwise distance $x_i + x_{i+1} + \cdots + x_{j-1}$ or the counterclockwise distance $x_j + x_{j+1} + \cdots + x_{i-1}$. If we knew the orientation of every edge, then we could write the isometry or anisometry constraint as a linear equation or nonequation. There are n choices for the orientations of the $n-1$ edges incident to a vertex p_i, depending on which wedge between consecutive edges contains the center of the circle. Considering all vertices, there are at most n^n choices for the orientations, each leading to a system of linear equations and nonequations. On the other hand, consider the n pairwise crossing edges $p_ip_{i+n/2}$; the orientation of each can be chosen independently. Hence, there are at least 2^n possible orientations.

For the edges that do not cross, the isometry and anisometry constraints can be reconstructed easily. Two edges p_ip_j and p_kp_l do not cross if their endpoints appear in the order p_i, p_j, p_k, p_l along the circumference. Fortunately, for constraints between such noncrossing edges, we can effectively determine the orientations of the edges to obtain a single system of linear equations and nonequations. To describe these constraints we first need the following lemma:

Lemma 1 *If the points p_i, p_j, p_k, p_l appear in clockwise order around the circle (with the possibility that $j = k$ or $l = i$ but not both) then we cannot have both edges p_ip_j and p_kp_l oriented counterclockwise.*

Proof. Suppose both edges are oriented counterclockwise. This implies that $\vec{d}(p_i, p_j) \geq \frac{1}{2}$ and $\vec{d}(p_k, p_l) \geq \frac{1}{2}$. Because of the ordering of the points around the circle, both p_k and p_l lie on the clockwise arc from p_j to p_i. Thus,

$$\vec{d}(p_j, p_i) = \vec{d}(p_j, p_k) + \vec{d}(p_k, p_l) + \vec{d}(p_l, p_i).$$

This equality cannot hold because $\vec{d}(p_j, p_i) \leq \frac{1}{2}$ while the right-hand-side is strictly greater than half. Thus, at least one of the edges must be oriented clockwise. □

We start with the isometry constraints, for which we need the following lemma:

Lemma 2 *If the points p_i, p_j, p_k, p_l appear in clockwise order around the circle (with the possibility that $j = k$ or $l = i$ but not both) such that $d(p_k, p_l) = d(p_i, p_j)$, then both edges p_ip_j and p_kp_l must be oriented clockwise.*

Proof. By Lemma 1 we know that we cannot orient both edges counterclockwise. Suppose only p_kp_l is oriented counterclockwise; this means that $d(p_k, p_l) = \vec{d}(p_l, p_k) \leq \frac{1}{2}$ and $d(p_i, p_j) = \vec{d}(p_i, p_j)$. Because of the ordering of the points p_i, p_j, p_k, p_l around the circle, we have

$$\vec{d}(p_l, p_k) = \vec{d}(p_l, p_i) + \vec{d}(p_i, p_j) + \vec{d}(p_j, p_k) \leq \frac{1}{2}.$$

Because at least one of the distances $\vec{d}(p_l, p_i)$ and $\vec{d}(p_j, p_k)$ is greater than zero, $\vec{d}(p_l, p_k) > \vec{d}(p_i, p_j)$ which implies that $d(p_k, p_l) > d(p_i, p_j)$; contradiction. Similarly, if only p_ip_j is oriented counterclockwise, then this would imply that $d(p_i, p_j) > d(p_k, p_l)$. Therefore, if $d(p_i, p_j) = d(p_k, p_l)$ then both edges must be oriented clockwise. □

Thus, if two edges p_ip_j and p_kp_l are noncrossing and their edge lengths must be equal, then by Lemma 2 we can assume that the edges are oriented clockwise and we force this by adding two linear inequalities:

$$\sum_{i \leq r < j} x_r \leq \frac{1}{2}, \quad \sum_{k \leq s < l} x_s \leq \frac{1}{2}.$$

Now that we have forced the edges to be oriented clockwise, we can force the equality of their lengths with a linear equation:

$$\sum_{i \leq r < j} x_r \;=\; \sum_{k \leq s < l} x_s.$$

For the anisometry constraints, we first show the following lemma:

Lemma 3 *If the points p_i, p_j, p_k, p_l appear in clockwise order around the circle (with the possibility that $j = k$ or $l = i$ but not both) such that at least one of the edges p_ip_j and p_kp_l must be oriented counterclockwise, then $\vec{d}(p_i, p_j) \neq \vec{d}(p_k, p_l)$.*

Proof. Assume that p_kp_l is oriented counterclockwise; then, $\vec{d}(p_l, p_k) \leq \frac{1}{2}$. By Lemma 1, p_ip_j must be oriented clockwise. Then, we have $\vec{d}(p_l, p_k) = \vec{d}(p_l, p_i) + \vec{d}(p_i, p_j) + \vec{d}(p_j, p_k) \leq \frac{1}{2}$. Because at least one of $\vec{d}(p_l, p_i)$ or $\vec{d}(p_j, p_k)$ is strictly positive, then $\vec{d}(p_k, p_l) > \vec{d}(p_i, p_j)$. Thus, the two edges have distinct clockwise lengths. □

Now consider an anisometry constraint between two noncrossing edges p_ip_j and p_kp_l. We represent this constraint by a linear nonequation:

$$\sum_{i \leq r < j} x_r \;\neq\; \sum_{k \leq s < l} x_s.$$

To show that the above nonequation holds independent of the orientation of each of the edges, we need to show that it holds if and only if the desired geodesic distances are distinct. First, if both edges can be oriented clockwise, then this constraint on the clockwise distances is equivalent to the distinctness of the geodesic lengths. Second, if one of the edges must be oriented counterclockwise, then by Lemma 2, the geodesic lengths must be different; and by Lemma 3, the linear nonequation must be satisfied.

4 Solving Systems of Linear Equations and Nonequations

The previous section yields a linear system of the following form:

$$Ax \leq \tfrac{1}{2} \tag{1}$$
$$Mx = f \tag{2}$$
$$Nx \neq g \tag{3}$$
$$x_i > 0 \tag{4}$$

Geometrically, (1), (2) and (4) define a (partly open) convex polyhedron P, while each row of (3) defines a forbidden hyperplane. To solve this system, we will use the following idea: initially we ignore the forbidden hyperplanes and find a relative interior point in the feasible region P. Then we check if this point lies on a forbidden hyperplane; if it does not lie on any forbidden hyperplane, we have a solution. Otherwise, if the point lies on h_i, then we recurse on one side of $h_i \cap P$. Recursing on one side of h_i ensures that we never pick a point on the same hyperplane more than once. Thus, in the worst case we will eventually run out of forbidden hyperplanes and one side of the last eliminated hyperplane will be a region with no forbidden points. We can now find a feasible point in this region.

After a suitable (and polynomial time computable) change of coordinates, we may assume that P has interior points. Let $h(a, \mu)$ denote the hyperplane $\{x \mid \langle a, x\rangle = \mu\}$.

Proposition 4 *If $h(a, \mu) \cap P$ has relative interior points, then for any sufficiently small $\varepsilon > 0$, $h(a, \mu+\varepsilon)$ and $h(a, \mu - \varepsilon)$ both have relative interior points.*

The idea is that given a point q in the relative interior of $P \cap h(a, \mu)$ (if such a q does not exist, then $h(a, \mu)$ is redundant and can be discarded), choose ε sufficiently small so that $h(a, \mu + \varepsilon) \cap P$ has relative interior, and does not collide with any possible parallel forbidden hyperplanes. If a solution exists, then there is one on the hyperplane $h(a, \mu + \varepsilon)$. To avoid calculation of ε, we add the inequality $a^T x > \mu$ to our system, replacing the nonequation $a^T x \neq \mu$. If this new system is infeasible, it means that $P \subset h(a, \mu)$, and the original problem is infeasible.

Our basic computational step is thus to find a relative interior point of a polyhedron defined by strict and nonstrict inequalities. This is equivalent to feasibility testing for systems of strict linear inequalities, which can be solved by linear programming.

The inequalities and (non)equations of our system describe the constraints for pairs of noncrossing edges. We have added a constant number of constraints per pair of noncrossing edges having the same or different labels. Because we have a total of $\binom{n}{2}$ edges, pairing them gives us $\Theta(n^4)$ constraints. The linear programs required can be solved in time polynomial in n and the number of bits required of the output. In our case, $\Theta(n)$ bits suffice to disambiguate all the distinct distances, at which point the solution will be correct; so we can solve the linear programs in time polynomial in n. The number of times we solve such a system is at most equal to the number of forbidden hyperplanes, which is a polynomial function of n. Thus we have a polynomial-time solution to the labeled beltway problem.

5 Open Problem

A natural variation on the labeled beltway problem treats the coloring as global instead of local: instead of just constraining the equality or inequality of the geodesic lengths of *incident* edges, it constrains the relative lengths of all edge pairs. This problem seems substantially more difficult because it becomes possible to construct specific numbers (less than 1) with $\Theta(n)$ bits. After some effort, it is not clear to us whether this global version of the labeled beltway problem is polynomially solvable or NP-complete.

Acknowledgments

This work was initiated at the 21st Bellairs Winter Workshop on Computational Geometry held January 27–February 3, 2005 at Bellairs Research Institute. We thank the other participants of that workshop for helpful discussions and contributing to a fun and creative atmosphere. Perouz also thanks Victor Campos for useful and entertaining discussions on the topic of this paper.

References

[1] E. D. Demaine, J. Erickson, D. Krizanc, H. Meijer, P. Morin, M. Overmars, and S. Whitesides. Realizing partitions respecting full and partial order information. *Journal of Discrete Algorithms*, 6:51–58, 2008.

[2] E. D. Demaine, F. Gomez-Martin, H. Meijer, D. Rappaport, P. Taslakian, G. T. Toussaint, T. Winograd, and D. R. Wood. The distance geometry of music. *Computational Geometry: Theory and Application*, 2008. To appear.

[3] P. Lemke, S. S. Skiena, and W. D. Smith. Reconstructing sets from interpoint distances. Technical Report 37, DIMACS Technical Report, 2002.

[4] R. D. Morris. New directions in the theory and analysis of musical contour. *Music Theory Spectrum*, 15(2):205–228, 1993.

[5] I. Quinn. Fuzzy extensions to the theory of contour. *Music Theory Spectrum*, 19(2):232–263, 1997.

[6] J. Rahn. Possible and impossible melodies: Some formal aspects of contour. *Journal of Music Theory*, 36(2):259–279, 1994.

[7] I. Shmulevich. A note on the pitch contour similarity index. *Journal of New Music Research*, 33(1):17–18, March 2004.

[8] S. S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56:275–294, 1994.

[9] G. T. Toussaint. A mathematical analysis of African, Brazilian, and Cuban *clave* rhythms. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 157–168, Towson University, Towson, Maryland, U.S.A., July 27-29 2002.

Guarding Orthogonal Art Galleries with Sliding Cameras

Matthew J. Katz [*] Gila Morgenstern[*]

Abstract

We study the problem of guarding a simple orthogonal art gallery P with security cameras sliding back and forth along straight tracks. We show that if only vertical (alternatively, horizontal) tracks are allowed, then a solution minimizing the number of tracks can be found in polynomial time. If both orientations are allowed, then a 3-approximation can be found in polynomial time, when a certain graph induced by P is perfect. In particular, if P is x-monotone (alternatively, y-monotone), then this graph is perfect and one can reduce the approximation factor to 2.

1 Introduction

A classical problem in computational geometry is the "art-gallery" problem, in which the goal is to determine how many points ("guards") are needed in order to see all parts of a geometric domain D (e.g., a simple polygon). During the years, many variants of the art-gallery problem have been studied. These variants differ in their assumptions concerning the underlying domain, the type of guards that may be employed, and the visibility model. Most of them are known to be NP-hard, including, e.g., the variant where the underlying domain is a simple orthogonal polygon and guards must lie at vertices; see, e.g., [7, 8].

Some of the variants, however, and especially those assuming a restricted model of visibility are solvable in polynomial time. In [5], Motwani et al. presented the "perfect graph approach," to solve an art-gallery problem under *s-visibility*. In this model, a guard located at a point p (of an orthogonal polygon P) sees all points that are connected to p by an orthogonal staircase path (contained in P). This approach was used later by Worman and Keil [9] to solve a similar problem under *r-visibility*. In this model, a guard located at a point p sees all points q for which the axis-parallel rectangle with diagonal $\overline{pq}$ is contained in P. In the perfect graph approach, first a graph G is associated with the given polygon P, and then the following two main claims are proven: (i) there is a one to one correspondence between a minimum guard cover of P and a minimum clique cover of G, and (ii) G is perfect. Note that the second claim is crucial, since, in general, minimum clique cover is NP-complete, but is polynomial for chordal or perfect graphs [2].

Many of the more recent variants involve mobile guards, where the requirement is that every point of the gallery is visible by some guard at some point along his path. Kay and Guay [3] gave an $O(n \log n)$ algorithm for the problem of determining whether a given polygon can be guarded by a single guard patrolling along a single line segment.

In this paper we consider the following problem. Let P be a simple orthogonal polygon. An orthogonal line segment $s \subseteq P$ is called a *segment guard* or a *seguard.* A seguard s *sees* a point $p \in P$ if there exists a point $q \in s$, such that the line segment $\overline{pq}$ is orthogonal and contained in P. We denote by $v(s)$ the region of P that is seen by a seguard s, and say that a seguard set S *guards* P if $\bigcup \{v(s) | \ s \in S\} = P$. In the minimum seguard cover problem the goal is to find such a set of minimum cardinality. One can think of a (e.g., horizontal) seguard s as a security camera sliding back and forth along a horizontal track and viewing, at every point along the track, directly upwards and directly downwards.

A *histogram polygon* is a simple polygon whose boundary consists of a base edge e and a chain that is monotone with respect to e. A *double-sided histogram polygon* is the union of two histogram polygons sharing the same base edge e and located on opposite sides of e. Fekete and Mitchell [1] proved that the histogram-decomposition problem, i.e., partitioning an orthogonal polygon P (with holes) into a minimum number of non-intersecting histogram polygons is NP-hard. For an orthogonal line segment $s \subseteq P$, denote by $H(s)$ the double-sided histogram polygon obtained by the infinite union of all maximal normals to s. The minimum seguard cover problem is equivalent to the problem of covering P with a minimum number of double-sided histogram polygons.

2 Preliminaries

We may assume that the endpoints of a seguard lie on P's boundary. Let $p \in P$ and let l_p^v (resp. l_p^h) denote the maximal vertical (resp. horizontal) line segment through p. Then, a seguard s sees p if and only if s crosses at least one of l_p^v and l_p^h.

Let $p, q \in P$. If one can draw a seguard that sees both p and q, we write $p \overline{\wedge} q$. Otherwise, we write $p \cancel{\overline{\wedge}} q$. Clearly, $p \overline{\wedge} q$ if and only if one can draw a seguard s, such that both p and q belong to $H(s)$.

For each reflex vertex of P, extend the two edges

[*]Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel; {matya, gilamor}@cs.bgu.ac.il. G. Morgenstern was partially supported by the Lynn and William Frankel Center for Computer Sciences.

adjacent to it until they hit P's boundary. (Thus, an edge between two reflex vertices is extended in both directions.) Let $S(P)$ denote the resulting set of extended edges together with the edges of P that were not extended. $S(P)$ induces a partition of P into rectangular regions; we denote these regions by $R(P)$.

It is easy to see that for any seguard s, there exists $s' \in S(P)$, such that $v(s) \subseteq v(s')$. Therefore, we will restrict ourselves to segments in $S(P)$.

It is also easy to see that if $p \in v(s)$, where $s \in S(P)$ and p is an internal point of a region r of $R(P)$, then $r \subseteq v(s)$. Therefore, if S is a set of seguards (i.e., a subset of $S(P)$), such that for each region r of $R(P)$, r's center point is seen by one or more of the seguards in S, then S is a guarding set of P.

Let $R_c(P)$ denote the set of center points of regions of $R(P)$. We conclude, that it is enough to find a subset of $S(P)$ of minimum cardinality, that collectively sees all points in $R_c(P)$.

3 Vertical Segment Cover

In this section we consider the minimum *vertical seguard cover* problem. Let P be a simple orthogonal polygon, find a set S of vertical seguards of minimum cardinality, such that $\bigcup_{s \in S} v(s) = P$.

In this section, the notation $\overline{\wedge}$ and $\not\overline{\wedge}$ refers only to vertical seguards. We associate with P the graph $G_v = \langle V, E \rangle$, where $V = R_c(P)$ and $(p,q) \in E$ if $p \overline{\wedge} q$. The major part of this section is devoted to proving Theorems 3 and 4 below, stating that (i) a minimum clique cover of G_v corresponds to a minimum vertical seguard cover of P, and (ii) G_v is chordal.

Let $p \in P$ and consider the double-sided histogram polygon $H(l_p^h)$. Notice that any maximal vertical segment contained in $H(l_p^h)$ sees p, and any vertical seguard that sees p is contained in $H(l_p^h)$. When thinking of $H(l_p^h)$ as the union of all maximal vertical seguards guarding p, we denote it by $\tilde{v}(p)$.

We shall need the following simple lemma.

Lemma 1 *Let $r \in P$ and let $p, q \in \tilde{v}(r)$ such that $\overline{pq}$ is horizontal and contained in P. Then $\overline{pq} \subseteq \tilde{v}(r)$.*

Proof. Assume, e.g., that $p_x < q_x$, and let $t \in \overline{pq}$. We show that $t \in \tilde{v}(r)$. Let p', q' and t' be the x-projections of p, q and t on l_r^h, and consider the rectangle R whose corners are at p, q, p', q'. Clearly, $R \subseteq P$ and in particular $\overline{tt'} \subseteq P$. The vertical line segment l_t^v intersects $\overline{p'q'}$ at t', that is, l_t^v intersects l_r^h at t', thus $l_t^v \subseteq \tilde{v}(r)$ and in particular $t \in \tilde{v}(r)$. □

We would like to prove that for any clique C of G_v, a single seguard is sufficient to guard all regions associated with the vertices of C. We use the following Helly-type theorem [4].

Theorem 2 [4] *If C is a family of simply-connected compact sets in the plane, such that any two members of C have a connected non-empty intersection and any three members of C have a non-empty intersection, then $\bigcap\{c \in C\}$ is non-empty and simply connected.*

Theorem 3 *A minimum clique cover of G_v corresponds to a minimum vertical seguard cover of P.*

Proof. Consider a vertical seguard cover of P. Then, clearly $v(s)$ is a clique of G_v, for each seguard s in the cover. Now, let $\mathcal{C}$ be a clique cover of G_v and let $C \in \mathcal{C}$. Below, we apply Theorem 2 to show that $\bigcap\{\tilde{v}(r) | r \in C\}$ is non-empty. Now, let $x \in \bigcap\{\tilde{v}(r) | r \in C\}$, then, by our observation just above Lemma 1, $l_x^v \subseteq \bigcap\{\tilde{v}(r) | \ r \in C\}$, that is, l_x^v is a vertical seguard guarding all regions associated with vertices of C.

It remains to show that $\bigcap\{\tilde{v}(r) | r \in C\}$ is non-empty. Let $p, q \in C$, then by definition $\tilde{v}(p) \cap \tilde{v}(q) \neq \emptyset$. Now, let $x, y \in \tilde{v}(p) \cap \tilde{v}(q)$. We show that there exists a simple path $\pi \subseteq \tilde{v}(p) \cap \tilde{v}(q)$ connecting x and y. Consider the line segments l_x^v and l_y^v. By our observation just above Lemma 1, $l_x^v, l_y^v \subseteq \tilde{v}(p) \cap \tilde{v}(q)$ (and both intersect l_p^h and l_q^h).

Denote by $\pi_1 \subseteq l_x^v$ (resp., $\pi_2 \subseteq l_y^v$) the vertical line segment between x and $l_x^v \cap l_p^h$ (resp., y and $l_y^v \cap l_p^h$). Finally, denote by π_3 the line segment between the points $\pi_1 \cap l_p^h$ and $\pi_2 \cap l_p^h$. By Lemma 1, $\pi_3 \subseteq \tilde{v}(p) \cap \tilde{v}(q)$. Now put $\pi = \pi_1 \pi_3 \pi_2$, then $\pi \subseteq \tilde{v}(p) \cap \tilde{v}(q)$ and connects x and y.

Let $p, q, r \in C$ and consider the line segments l_p^h, l_q^h and l_r^h. Let s_{pq} be a vertical seguard guarding both p and q, and thus intersecting both l_p^h and l_q^h. Similarly, s_{pr} intersects both l_p^h and l_r^h, and s_{qr} intersects both l_q^h and l_r^h. Therefore, at least one of the three guards must intersect all three segments l_p^h, l_q^h, l_r^h, and is therefore contained in $\tilde{v}(p) \cap \tilde{v}(q) \cap \tilde{v}(r)$. □

Theorem 3 guarantees that finding a minimum clique cover of G_v is sufficient. Our next goal is to prove Theorem 4 that states that G_v is chordal. Before then, note that the border(s) between $\tilde{v}(p)$ and $P \setminus \tilde{v}(p)$, assuming $\tilde{v}(p) \neq P$, are vertical line segments. This implies Observation 1 below.

Observation 1 *Let $p, q \in P$, such that $\tilde{v}(p) \cap \tilde{v}(q) \neq \emptyset$ and neither $\tilde{v}(p) \subseteq \tilde{v}(q)$ nor $\tilde{v}(q) \subseteq \tilde{v}(p)$. Then, (i) the region $P \setminus (\tilde{v}(p) \cap \tilde{v}(q))$ is not connected, and (ii) any connected component of $\tilde{v}(p) \setminus \tilde{v}(q)$ and any connected component of $\tilde{v}(q) \setminus \tilde{v}(p)$ are contained in two different connected components of $P \setminus (\tilde{v}(p) \cap \tilde{v}(q))$.*

Theorem 4 *G_v is chordal.*

Proof. We have to show that there is no hole of size $k \geq 4$ in G_v. Let H be a cycle in G_v of size $k \geq 4$. We show that there must exist a chord in H. If there exist $p, q, r \in H$, such that $\tilde{v}(p) \cap \tilde{v}(q) \cap \tilde{v}(r) \neq \emptyset$, then we are done. Otherwise, each adjacent pair $p, q \in H$ satisfies the conditions of Observation 1. Therefore, there exists a vertex in H with only one neighbor in H, a contradiction. $\square$

Theorem 5 *Let P be an orthogonal n-gon. Then, an exact minimum vertical seguard cover of P can be found in time polynomial in n.* $\square$

4 Orthogonal Segment Cover

In the orthogonal version of segment cover, we may employ horizontal as well as vertical seguards. We associate with P the graph $G = \langle V, E \rangle$, where $V = R_c(P)$ and $(p, q) \in E$ if $p \barwedge q$. (Note that unlike G_v, two regions in $R_c(P)$ are adjacent in G if there exists a vertical or horizontal segment that guards both regions.) Unfortunately, Theorem 3 does not hold in the orthogonal version. Figure 1(a) shows a clique polygon (in which any two regions can be guarded by a single seguard), that cannot be guarded by a single seguard. However, three seguards are sufficient in this case, as is proven in Lemma 7 below.

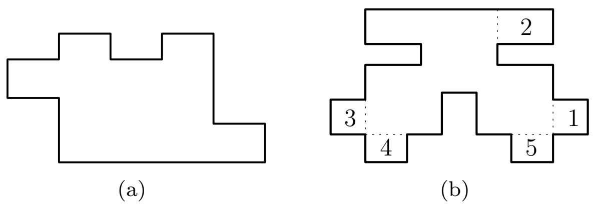


Figure 1: (a) Any two regions can be guarded by a single seguard, but there is no seguard that guards all regions. (b) Regions 1,2,3,4,5 form a chordless cycle of size 5.

For $p, q \in P$ and $s \in S(P)$, let $\delta(p, q)$ denote the number of links in a minimum-link orthogonal path between p and q (that is contained in P), and let $\delta(p, s) = \min_{q \in s} \delta(p, q)$. If s sees p, then clearly $\delta(p, s) \leq 1$, moreover if $p \barwedge q$ then $\delta(p, q) \leq 3$. Thus, if C is a subpolygon of P corresponding to a clique of G, then for each $p, q \in C$ we have that $\delta(p, q) \leq 3$.

Lemma 6 *[6] Let P be an orthogonal polygon, such that for any $p, q \in P$, $\delta(p, q) \leq k$. Then, there exists an orthogonal line segment $s \subseteq P$, such that for any $p \in P$, $\lfloor \frac{k}{2} \rfloor \leq \delta(p, s) \leq \lfloor \frac{k}{2} \rfloor + 1$.*

Lemma 7 *Let C be a subpolygon of P corresponding to a clique of G, then three seguards are sufficient to guard C.*

Proof. By Lemma 6, there exists a line segment $s \subseteq C$, such that for any $p \in C$, $\delta(p, s) \leq 2$. Assume, e.g., that s is vertical. If for any $p \in P$, $\delta(p, s) \leq 1$, then we are done. Assume therefore that $v(s) \subsetneq C$, and let $p \in C$ be a point for which $\delta(p, s) = 2$. Notice that any 2-link orthogonal path connecting p and s must start with a vertical line segment. We claim that there cannot exist two such paths where in one the vertical link is y-increasing and in the other the vertical link is y-decreasing. Assume there exist such paths $\pi_\uparrow$ which is y-increasing and $\pi_\downarrow$ which is y-decreasing, and consider the rectangle R bounded by $\pi_\uparrow$, $\pi_\downarrow$ and s. Clearly, $R \subseteq C$ and s sees p through R, thus $\delta(p, s) = 1$, a contradiction.

Let $P_\uparrow \subseteq C$ (resp. $P_\downarrow \subseteq C$) be the set of all points $p \in C$, for which $\delta(p, s) = 2$ and the 2-link paths to s are y-increasing (resp. y-decreasing). For $p \in P_\uparrow$, let $p_\uparrow$ be the highest point on s at which a 2-link path from p can end. Let $q \in P_\uparrow$ such that for any $p \in P_\uparrow$, $q_\uparrow$ is not above $p_\uparrow$. We denote by $s_\uparrow \subseteq C$ the horizontal line segment through $q_\uparrow$. The horizontal line segment $s_\downarrow$ is defined analogously. Note that $P_\uparrow$ (resp. $P_\downarrow$) might be empty, in which case $s_\uparrow$ (resp. $s_\downarrow$) is not defined.

We now claim that for any $p \in P_\uparrow$, p is seen from $s_\uparrow$, and, similarly, for any $p \in P_\downarrow$, p is seen from $s_\downarrow$. This will imply that the three seguards s, $s_\uparrow$ and $s_\downarrow$ together guard C.

Indeed, let $p \in P_\uparrow$. $s_\uparrow$ divides C into two subpolygons. We call the subpolygon containing the points directly below $s_\uparrow$ the lower subpolygon, and the subpolygon containing the points directly above $s_\uparrow$ the upper subpolygon. Assume first that p lies in the lower subpolygon. Since $p_\uparrow$ lies in the upper subpolygon, the vertical link of the 2-link path from p to $p_\uparrow$ must cross $s_\uparrow$, implying that $s_\uparrow$ sees p.

Now assume that p lies in the upper subpolygon. We show that this is impossible. Notice that since $p \in P_\uparrow$, the minimum-link path from p to $s_\uparrow$ in this case consists of exactly three links. We show that this implies $\delta(p, q) > 3$. Let q' be q's projection onto $s_\uparrow$. If $\delta(p, q') = 3$, then there exists a 2-link y-increasing path connecting q and l that meets l at a point above $q_\uparrow$, which is impossible by our construction. Otherwise, $\delta(p, q) = \delta(p, s_\uparrow) + 1 + \delta(q', q) = 3 + 1 + 1 = 5 > 3$, a contradiction.

Similarly, we show that for any $p \in P_\downarrow$, p is seen from $s_\downarrow$. $\square$

Theorem 4 does not hold as well in the orthogonal version. Figure 1(b) shows that, in general, G is not even perfect. However, for some important families of polygons, G is perfect. One such family is the family of x-monotone (alternatively, y-monotone) polygons (see Theorem 10). If P is x-monotone, then it is not difficult to see that two guards are sufficient in order to guard a subpolygon corresponding to a clique of G. Actually, we believe that two guards are always sufficient to guard such subpolygons.

Let P be a x-monotone polygon. Before proving Theorem 10 that states that G in this case is perfect, we make a few observations.

Lemma 8 *Let $u, v, w \in P$ be points, such that $u \barwedge v$, $w \not\barwedge u$ and $w \not\barwedge v$, then either $w_x < u_x, v_x$ or $u_x, v_x < w_x$.*

Proof. Assume, e.g., that $u_x \leq v_x$, and assume to the contrary that $u_x \leq w_x \leq v_x$. Let s_{uv} be a seguard guarding both u and v. If s_{uv} is horizontal, then it clearly also sees w. Let w' be the x-projection of w onto s_{uv}. Since P is x-monotone, $\overline{ww'} \subseteq P$, and s_{uv} sees w from w', implying that $w \barwedge v$ (as well as $w \barwedge u$), a contradiction.

Otherwise, s_{uv} is vertical and assume, e.g., that w is to the right of s_{uv} (if w is on s_{uv}, then clearly w is seen by s_{uv}). $\overline{vv'} \subseteq P$, where v' is the y-projection of v onto s_{uv}, since s_{uv} sees v. Note that $v'_x < w_x \leq v_x$ and let w' be the x-projection of w onto $\overline{vv'}$. Again, since P is x-monotone, $\overline{ww'} \subseteq P$. Moreover $\overline{vw'} \subseteq P$, therefore l_w^v (as well as l_v^h) see both v and w, implying that $w \barwedge v$, a contradiction. $\square$

Let H be a hole or anti-hole in G, and let π_H denote some permutation of the vertices of H in which $\pi_H(u) \leq \pi_H(v)$ if and only if $u_x \leq v_x$, where u, v are any two vertices in H. The following observations on π_H follow from Lemma 8.

Observation 2 *Let $H = \langle 1, 2, 3, \cdots, k\rangle$ be an anti-hole of length $k \geq 5$. Then, for any three consecutive vertices u, w, v of H, we have $u \barwedge v$, $w \not\barwedge u$ and $w \not\barwedge v$. Therefore, by Lemma 8, w cannot appear between u and v in π_H. In other words, the following subsequences and their reverse subsequences cannot appear in π_H: $\langle 1,2,3\rangle, \langle 2,3,4\rangle, \cdots, \langle k-1,k,1\rangle, \langle k,1,2\rangle$.*

Observation 3 *Let $H = \langle 1, 2, 3, \cdots, k\rangle$ be a hole of length $k \geq 5$. Then the following subsequences and their reverse subsequences cannot appear in π_H: $\langle 1,4,2\rangle, \langle 1,5,2\rangle, \cdots, \langle 1,k-1,2\rangle, \langle 2,5,3\rangle, \langle 2,6,3\rangle, \cdots, \langle 2,k,3\rangle, \cdots\cdots, \langle k,3,1\rangle, \langle k,4,1\rangle, \cdots, \langle k,k-2,1\rangle$.*

Lemma 9 *Let $n \geq 4$ and let $3 \leq k \leq n$. Let π_n be a permutation of $\{1, 2, \ldots, n\}$, such that (i) $\pi_n(1) = 1$ (i.e., the number at place 1 of π_n is 1), and (ii) for each $1 \leq i \leq k-2$, $\langle i,i+1,i+2\rangle$ and $\langle i+2,i+1,i\rangle$ are no subsequences of π_n. Then: $\pi_n^{-1}(k) < \pi_n^{-1}(k-1)$ if k is odd, and $\pi_n^{-1}(k-1) < \pi_n^{-1}(k)$ otherwise.*

Proof. We prove the lemma by induction on k.
$k=3$: As $\pi_n(1) = 1$, we have $\pi_n^{-1}(2) > 1$. Now, since $\langle 1,2,3\rangle$ is not a subsequence of π_n, we conclude that $\pi_n^{-1}(3) < \pi_n^{-1}(2)$.
$k=4$: As above, $\pi_n(1) = 1$ and $\pi_n^{-1}(3) < \pi_n^{-1}(2)$. Now, since $\langle 4,3,2\rangle$ is not a subsequence of π_n, we conclude that $\pi_n^{-1}(3) < \pi_n^{-1}(4)$.

Assume the lemma is true for $k-1 < n$. If k is odd, then $k-1$ is even and, by the induction hypothesis, we have that $\pi_n^{-1}(k-2) < \pi_n^{-1}(k-1)$. Now, since $\langle k-2,k-1,k\rangle$ is not a subsequence of π_n, we conclude that $\pi_n^{-1}(k) < \pi_n^{-1}(k-1)$.

If k is even, then $k-1$ is odd and, by the induction hypothesis, we have that $\pi_n^{-1}(k-1) < \pi_n^{-1}(k-2)$. Now, since $\langle k,k-1,k-2\rangle$ is not a subsequence of π_n, we conclude that $\pi_n^{-1}(k-1) < \pi_n^{-1}(k)$. $\square$

Theorem 10 *G is perfect.*

Proof. We need to show that G does not contain an odd hole or an odd anti-hole of size 5 or greater.

Assume to the contrary that G contains an odd hole $H = \langle 1, 2, 3, \cdots, k\rangle$, where $k \geq 5$. Consider π_H and assume w.l.o.g. that $\pi_H(1) = 1$. Clearly $\pi_H^{-1}(2) > 1$. By Observation 3, $\langle 1,k-1,2\rangle$ is not a subsequence of π_H, and therefore 2 must precede $k-1$ in π_H, i.e., $\pi_H^{-1}(2) < \pi_H^{-1}(k-1)$. By the same observation, $\langle k,2,k-1\rangle$ is not a subsequence of π_H, implying that $\pi_c^{-1}(2) < \pi_c^{-1}(k)$, and $\langle 1,3,k\rangle$ is also not a subsequence of π_H, implying that $\pi_c^{-1}(k) < \pi_c^{-1}(3)$. We get that $\langle 2,k,3\rangle$ is a subsequence of π_H, a contradiction.

Now, assume to the contrary that G contains an odd anti-hole H of size $k \geq 5$. Consider π_H and assume w.l.o.g. that $\pi_H(1) = 1$. By Observation 2, neither $\langle i,i+1,i+2\rangle$ nor $\langle i+2,i+1,i\rangle$ is a subsequence of π_H, for $i = 1, \ldots, k-2$, and therefore, by Lemma 9, $\pi_H^{-1}(k) < \pi_H^{-1}(k-1)$. We get that $\langle 1,k,k-1\rangle$ is a subsequence of π_H, a contradiction. $\square$

Theorem 11 *Let P be an orthogonal n-gon. Then, a 2-approximation for orthogonal seguard cover of P can be found in time polynomial in n, if P is x-monotone, and a 3-approximation can be found, if P belongs to another family of polygons for which G is perfect.* $\square$

References

[1] S. P. Fekete and J. S. B. Mitchell. Terrain decomposition and layered manufacturing. *Int. J. Comput. Geom. Appl.*, 11:647–668, 2001.

[2] M. Grötschel, L. Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. In C. Berge and V. Chvátal, editors, *Topics on perfect graphs*, volume 21 of *Annals of Discrete Mathematics*, pages 325–356. North-Holland, 1984.

[3] D. Kay and M. Guay. Convexity and a certain property p_m. *Israel Journal of Mathematics*, 8:39–52, 1970.

[4] J. Molnár. Über den zweidimensionalen topologischen satz von Helly. *Mat. Lapok*, 8:108–114, 1957.

[5] R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. In *Proc. 4th Sympos. Comput. Geom.*, pages 211–223, 1988.

[6] B. J. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a rectilinear polygon. *Comput. Geom. Theory Appl.*, 6(3):169–194, 1996.

[7] J. O'Rourke. *Art Gallery Theorems and Algorithms.* The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[8] D. Schuchardt and H.-D. Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Q.*, 41:261–267, 1995.

[9] C. Worman and J. M. Keil. Polygon decomposition and the orthogonal art gallery problem. *Int. J. Comput. Geom. Appl.*, 17(2):105–138, 2007.

Inspecting a Set of Strips Optimally

Tom Kamphans* Elmar Langetepe**

Abstract

We consider a set of axis-parallel nonintersecting strips in the plane. An observer starts to the left of all strips and ends to the right, thus visiting all strips in the given order. A strip is *inspected* as long as the observer is inside the strip. How should the observer move to inspect the set of strips?

Keywords: Motion planning, watchman routes

1 Introduction

In the last decades, routes from which an agent can see every point in a given environment have drawn a lot of attention (e.g., [3, 4, 5, 6]). Usually, the objective is to find a short route; either the shortest possible route (the optimum) or an approximation. In this paper, we focus on another criterion for routes: We want to minimize the time that a certain area of the environment is *not* seen. Imagine a guard in an art gallery whose objective is to be as vigilant as possible and to minimize the time an object is unguarded. We restrict ourselves to a very simple kind of environments—parallel strips in the plane. More complicated environments are the subject of ongoing research. In Section 2 we present notational conventions and define an objective function which has to be minimized. Then, in Section 3 we first prove some structural properties of an optimal solution for the Euclidean case. At the end we present an efficient algorithm. The ideas can be adapted to the L_1-case which is mentioned in Section 4.

2 Preliminaries

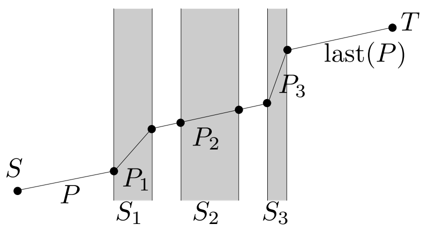


Figure 1: Visiting three strips in a given order.

*Braunschweig University of Technology, Computer Science, Algorithms Group, 38106 Braunschweig, Germany

**University of Bonn, Institute of Computer Science I, 53117 Bonn, Germany

Let $\{S_1, \ldots, S_n\}$ be a set of nonintersecting vertical strips and $S = (s_x, s_y)$ be a start point to the left of all strips and $T = (t_x, t_y)$ be an end point to the right of all strips. W.l.o.g. we can assume that S is below T (i.e., $s_y \leq t_y$). Strip S_i has width w_i.

An inspection path, P, from S to T visits the strips successively from left to right, see Fig. 1. For a given path P let P_i denote the part of P within strip S_i. Let $|P_i|$ denote the corresponding path length, and last(P) the last segment of P (i.e., from S_n to T).

While P visits S_i, the strip is entirely visible. The performance of P for a single strip S_i therefore is given by $\mathrm{Perf}(P, P_i) := |P| - |P_i|$. The performance of the path P for all strips is given by the worst performance achieved for a single strip. That is $\mathrm{Perf}(P) := \max_i \mathrm{Perf}(P, P_i)$. Finally, the task is to find among all inspection paths the path that gives the best performance for the given situation; that is, an inspection path with minimal performance:
$\mathrm{Perf} := \min_P \max_i \mathrm{Perf}(P, P_i)$.

This problem belongs to the class of LP-type problems [7], but the basis could have size n. Therefore, we solve the problem directly. It may also be seen as a *Time and Space* Problem (see, e.g., [2, 1]).

3 The Euclidean Case

In this section we first collect some properties of the optimal solution and design an efficient algorithm.

3.1 Structural Properties

We can assume that the optimal inspection path is a polygonal chain with straight line segments inside and between the strips: If there are kinks or arcs inside or outside the strips, we can optimize the inspection path by straightening the corresponding parts.

Let us further assume that we have an inspection path as depicted in Fig. 2. The first simple observation is that we can rearrange the set of strips in any nonintersecting order and combine the elements of the given path adequately by shifting the segments horizontally without changing the path length.

Now, it is easy to see that an optimal solution has the same slope between all strips. We rearrange the strips such that they stick together and start from the X-coordinate of the start point. The last part of the solution should have no kinks as mentioned earlier.

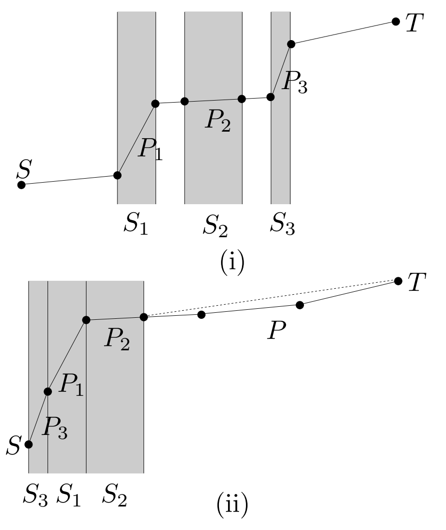


Figure 2: Rearranging strips and path yields the same objective value. Thus, the optimal solution has the same slope between all strips.

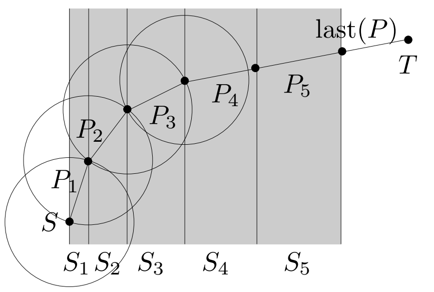


Figure 3: An optimal solution: The first strips are visited with $|P_i| = d$, every other strip with $|P_i| > d$.

Lemma 1 *The optimal solution is a polygonal chain without kinks between the strips or inside the strips. The path has the same slope between all strips.*

In the following, we assume that the strips are ordered by widths $w_1 \leq w_2 \leq \cdots \leq w_n$, starting at the X-coordinate of the start point s_x, and lie side by side (i.e., without overlaps or gaps), see Fig. 2(ii). For $t_y = s_y$ the optimal path is the horizontal connection between S and T. Thus, we assume $t_y > s_y$.

Now, we show some structural properties of an optimal solution. The optimal solution visits some strips with the same value $d = |P_i|$ until it finally moves directly to the end point, see Fig. 3 for an example.

Lemma 2 *In a setting as described earlier, the optimal path P visits the first $k \leq n$ strips with the same distance d and then moves directly to the endpoint. That is, for $i = 1, \ldots, k$ we have $|P_i| = d$ and for $i = k+1, \ldots, n$ we have $|P_i| > d$. The path is monotonically increasing and convex with respect to the segment ST.*

Proof. Let P denote an optimal path for n strips. First, we show that the path is monotonically increasing. There is at least one segment, P_i, with positive slope, because $t_y > s_y$. Let us assume—for contradiction—that there is also a segment, P_j, with negative slope. We rearrange the strips and the path such that P_i immediately succeeds P_j, see Fig. 4(i). Now, we can move the common point of P_j and P_i upwards. Both segments decrease and the performance of P gets better. Thus, there is no segment with negative slope and P is monotonically increasing.

The performance of P is given by $|P_k| := \min_j |P_j|$. Let k be the biggest index such that P_k is responsible for the performance. By contradiction, we show that for $i < k$ there is no P_i with $|P_i| > |P_k|$. So let us assume that $|P_i| > |P_k|$ holds for $i < k$. We can assume $|P_k| = |P_{i+1}|$. From $w_i \leq w_{i+1}$ we conclude that the path P_iP_{i+1} makes a right turn. Because $|P_i| > |P_k|$ we can globally optimize the solution by moving the connection point downward, see Fig. 4(ii). Although P_i increases, the total path length decreases. Thus, $|P_i| = |P_k|$ for $i = 1, \ldots, k-1$. For $i = k+1, \ldots, n$ we have $|P_i| > |P_k|$ by assumption.

Now, we show that there is no kink in the path $P_{k+1}P_{k+2}\cdots P_n$. As $|P_j| > |P_k|$ and $|P_{j+1}| > |P_k|$ we can globally optimize the solution by moving the connection point downwards or upwards, see Fig. 4(iii). The path length decreases. Thus, $P_{k+1}P_{k+2}\cdots P_n$ is a straight line segment.

Altogether, for $i = k+1, \ldots, n$ we have $|P_i| > d$. For $i = 1, \ldots, k$ we have $|P_i| = d$ and the part $P_{k+1}P_{k+2}\cdots P_n\text{last}(P)$ is a straight line segment. Path P is monotonically increasing. The first part of P makes only right turns as already seen. The last part is a line segment. The concatenation of P_k and $P_{k+1}P_{k+2}\cdots P_n$ also makes a right turn; otherwise, we can again improve the performance because $P_{k+1} > P_k$. Altogether, P is convex w.r.t. ST. □

In the following, we show that we can compute the optimal solution incrementally; that is, we successively add new strips and consider the corresponding optimal solutions.

Let $S_1, S_2, \ldots, S_n$ be a set of strips and let S and T be fixed. Let P^i denote the optimal solution for the first i strips. For increasing i the parameter k of Lemma 2 is strictly increasing until it remains fixed:

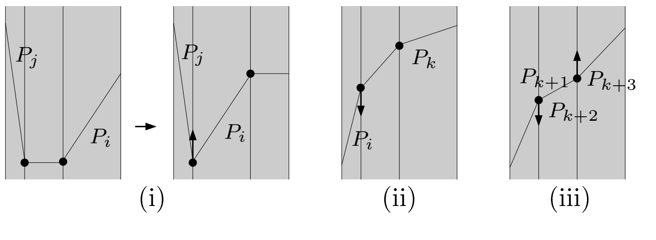


Figure 4: Global optimization by local changes: The connection point can be moved (i) upwards, (ii) downwards or (iii) upwards or downwards.

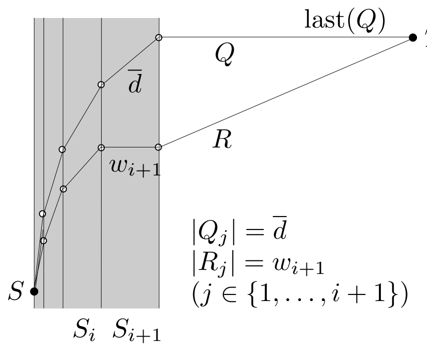


Figure 5: An optimal solution for $k = i + 1$ strips can be found between the extremes R and Q. $|R_j| = w_{i+1}$ holds and $\overline{d}$ fulfills $|Q_j| = \overline{d}$ with horizontal last(Q).

Lemma 3 *For P^i either the index k is equal to i or P^i is already given by P^{i-1}. If P^i is identical to P^{i-1}, also P^j is identical to P^{i-1} for $j = i+1,\dots,n$.*

We postpone the complete proof of Lemma 3 and first show how to adapt a solution P^i to a solution P^{i+1}. Let us assume that we have a solution P^i and that k in the sense of Lemma 2 is equal to i. That is, the first i strips are visited with the same distance d. If $|P^i_{i+1}| < d$ holds, the solution P^i is not optimal for $i+1$ strips. Therefore we want to adapt P^i. Lemma 3 states that it migth be useful to search for a solution with identical path length in $k = i+1$ strips.

We will now show that this solution can be computed efficiently. The task is to compute the path P^{i+1} between two extreme solutions, R and Q, as follows: Let R be the path with path lengths $|R_j| = w_{i+1}$ for $j = 1,\dots,i+1$ and let Q be the path with $|Q_j| = \overline{d}$ for $j = 1,\dots,i+1$, where the last segment, last(Q), is horizontal. Starting from $d = w_{i+1}$, let $P^{i+1}(d)$ denote the unique path that starts with $|P^{i+1}_j| = d$ for $j = 1,\dots,i+1$ and ends with a straight line segment.

The performance of $P^{i+1}(d)$ is given by the function $f_{i+1}(d) := di + |\text{last}(P^{i+1}(d))|$. Note that we can express $|\text{last}(P^j(d))|$ in terms of d: $|\text{last}(P^{i+1}(d))| = \sqrt{X^2 + \left(t_y - \sum_{j=1}^{i+1}\sqrt{d^2 - w_j^2}\right)^2}$ with $X := t_x - \sum_{j=1}^{i+1} w_j$. By simple analysis, we can show that $f_{i+1}(d)$ has a unique minimum in d:

Lemma 4 *The function $f_{i+1}(d)$ has exactly one minimum for $d \in [w_{i+1}, \delta]$, where δ is the solution of $t_y - \sum_{j=1}^{i+1}\sqrt{\delta^2 - w_j^2} = 0$ (i.e., the last segment is horizontal).*

Now, it is easy to successively compute the minimum of $f_{i+1}(d)$ for $i = 0,\dots,n-1$. For example, we can apply numerical methods for getting a solution of $f'_{i+1}(d) = 0$. In the following we assume that we can compute this minimum in time $O(i)$.

Using Lemma 4 we can now prove Lemma 3.

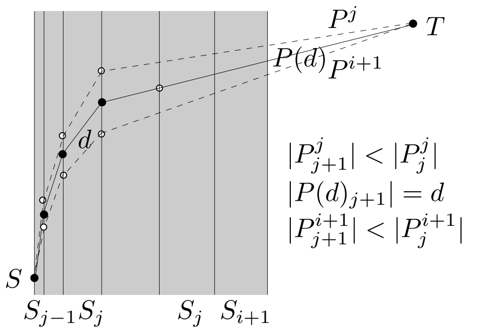


Figure 6: Between P^{i+1} and P^j there has to be a solution with $P(d)$ which is better than P^{i+1}.

Proof of Lemma 3. Let us assume that we have a solution P^i and let $|P^i_k| = d$, where k denotes the index in the sense of Lemma 2. We use this solution for the first $i+1$ strips. If $|P^i_{i+1}| \geq d$ holds, the given solution is also optimal for $i+1$ strips because the overall performance remains the same: The last segment, last(P^i), of P^i is a line segment with positive slope. Further, $w_j \geq w_{j-1}$ holds. Thus, P^i is the overall optimum; that is, we can apply P^i to all n strips and $|P^i_j| \geq d$ holds for $j = i+1, i+2, n$.

This means that, if we have found a solution P^i with $|P^i_{i+1}| \geq |P^i_k|$ where k denotes the index from Lemma 2, then we are done for all strips.

It remains to show that the index k is strictly increasing until it is finally fixed. From the consideration above we already conclude that there is only one strip, where k does not increase. If k does not increase from i to $i+1$ we have $k < i+1$ and $|P^{i+1}_k| < |P^{i+1}_{i+1}|$. Thus, P^{i+1} is the overall optimum and k is fixed.

Finally, we show that indeed k can never decrease. Let us assume from $\ell = 1$ to $\ell = i$ we have a solution P^ℓ for ℓ strips and $k = \ell$ for every P^ℓ. Let us further assume that for $i+1$ strips the solution P^{i+1} comes along with $k = j < i$. We compare the two solutions P^{i+1} (with $k = j < i$ and $|P^{i+1}_j| =: d_{i+1}$) and P^j (with $k = j$ and $|P^j_j| =: d_j$), see Fig. 6.

As P^j is optimal for j strips but not for $j+1$, we have $|P^j_{j+1}| < d_j$. On the other hand P^{i+1} is optimal for $i+1$ strips; thus $|P^{i+1}_{j+1}| > d_{i+1}$ holds; see Lemma 2.

Now for a parameter d consider a monotone path $P(d)$ that starts from S, has equal path length d in the first j strips and then moves toward T. While d increases, the slope of the last segment strictly decreases. Therefore, the path length $|P(d)_{j+1}|$ is strictly decreasing in d. This means that $d_j > d_{i+1}$ and P^j runs above P^{i+1}. The path $P(d)$ changes continuously, therefore in $[d_{i+1}, d_j]$ there has to be a value d' such that $|P(d')_{j+1}|$ is equal to d'. The path $P(d')$ runs between P^{i+1} and P^j. Obviously, $d' < P(d')_l$ for $l = j+1,\dots,n$ holds.

We show that $P(d')$ is better than P^{i+1}. This is a direct consequence of Lemma 4. The value of $P(d)$ strictly decreases from $d = d_{i+1}$ to the unique minimum $d = d_j$. Altogether, P^{i+1} is not optimal. $\square$

The result of Lemma 3 now suggests a method for computing the optimal path efficiently. Starting from $j = 1$ we compute an optimal path for the first j strips. Let P^j denote this path. If $|P^j_{j+1}| > |P^j_j|$ holds we are done. Otherwise, we have to compute P^{j+1} for $j+1$ strips and so on.

3.2 Algorithm and its Analysis

Theorem 5 *For a set of n axis-aligned strips the optimal inspection path can be computed in $O(n \log n)$ time and linear space.*

Proof. First, we sort the strips by width which takes $O(n \log n)$ time. Then we apply binary search. That is, in a first step we compute a solution with respect to $j = \lfloor \frac{n}{2} \rfloor$ strips. This can be done by computing the best value for $f_j(d)$ starting from $w_j = d$ until the last segment is horizontal, see Lemma 4. Let d_j denote the optimal value for j strips and P^j the optimal path.

Now, we have to determine whether the optimal path visits $i \leq j$ or $i > j$ strips with the same distance d_i. If $d_j > w_{j+1}$, we have to take into account at least the strip S_{j+1}. Therefore, $i > j$ holds and we proceed recursively with the interval $[j, n]$. If $d_j \leq x_{j+1}$ we proceed with the interval $[1, j]$. Therefore we will find the optimum in $\log n$ steps. Computing the minimum of $f_j(d)$ for index j takes $O(j)$ time. $\square$

For a lower bound construction we can simply assume that the input of an algorithm is given by an unsorted set of strips. The X-coordinates of the strip's left boundaries and their widths describe the setting. The solution is given by a polygonal chain from left to right representing the order of the left boundaries. Thus, sorting a set of n elements can be reduced to the given problem.

Theorem 6 *For a set of n axis-aligned unsorted strips the optimal inspection path is computed in $\Theta(n \log n)$ time and $\Theta(n)$ space.*

4 The L_1-Case

Fortunately, if we measure the distance by the L_1 metric the structural properties are equivalent. Computing the optimal path becomes much easier.

Note that the path segments between the strips have to be horizontal. We have to distribute the vertical distance from S to T among a subset of the strips. Again we sort the strips by their widths and rearrange the scenario. Fig. 7 shows an example of an optimal L_1-path after rearrangement.

Theorem 7 *The optimal L_1-path P visits the first $k \leq n$ strips with the same L_1-distance d and then moves horizontally to the end point T. For $i = 1, \ldots, k$ we have $|P_i| = d$ and for $i = k+1, \ldots, n$ we have $w_i > d$. Additionally, $\sum_{i=1}^{k} |P_i| = t_y$ holds. If the number of strips, n, increases, the index k increases until it remains fixed. The optimal path can be computed in $\Theta(n \log n)$ time and $\Theta(n)$ space.*

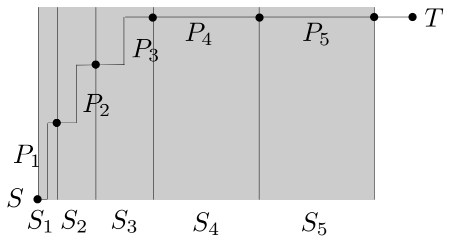


Figure 7: An optimal solution in the L_1-case after rearrangement. The first three strips are visited with the same value d, for all other strips $|P_i| > d$ holds. The path is horizontal between strips.

An algorithm for the L_1 problem is given as follows. First, we sort the strips by their widths. Then starting from $i = 1$ we distribute $t_y + \sum_{j=1}^{i} w_j$ among i strips. For an optimal path, P^i, for i strips we have $|P^i_j| = \frac{1}{i}(t_y + \sum_{j=1}^{i} w_j)$ for $j = 1, \ldots, i$. If $|P^i_i| < w_{i+1}$ this path is also optimal for $i+1$ (and n) strips. For $|P^i_i| > w_{i+1}$ we distribute $t_y + \sum_{j=1}^{i+1} w_j$ among $i+1$ strips; that is, $|P^{i+1}_j| = \frac{1}{i+1}(t_y + \sum_{j=1}^{i+1} w_j)$ for $j = 1, \ldots, i+1$.

Altogether, if the strips are given by ordered widths, the algorithm runs in $\Theta(n)$ time and space.

References

[1] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. In *Proc. 18th Internat. Sympos. Algorithms Comput.*, volume 4835 of *Lecture Notes Comput. Sci.*, pages 776–787, 2007.

[2] F. Berger, A. Grüne, and R. Klein. How many lions can one man avoid? Technical Report 006, Department of Computer Science I, University of Bonn, 2007. http://web.informatik.uni-bonn.de/I/publications/bgk-hmlcm-07.pdf.

[3] W. Chin and S. Ntafos. Optimum watchman routes. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 24–33, 1986.

[4] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 473–482, 2003.

[5] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM J. Comput.*, 31:577–600, 2001.

[6] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. In H. Bunke, H. I. Christensen, G. D. Hager, and R. Klein, editors, *Sensor Based Intelligent Robots*, volume 2238 of *Lecture Notes Comput. Sci.*, pages 245–258, Berlin, 2002. Springer.

[7] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.*, volume 577 of *Lecture Notes Comput. Sci.*, pages 569–579. Springer-Verlag, 1992.

Modem Illumination of Monotone Polygons

Oswin Aichholzer[*§] Ruy Fabila-Monroy[†] David Flores-Peñaloza[†] Thomas Hackl[*]
Clemens Huemer [‡] Jorge Urrutia[†¶] Birgit Vogtenhuber[*]

Abstract

We study a generalization of the classical problem of illumination of polygons. Instead of modeling a light source we model a wireless device whose radio signal can penetrate a given number k of walls. We call these objects k-modems and study the minimum number of k-modems necessary to illuminate monotone and monotone orthogonal polygons. We show that every monotone polygon on n vertices can be illuminated with $\left\lceil \frac{n}{2k} \right\rceil$ k-modems and exhibit examples of monotone polygons requiring $\left\lceil \frac{n}{2k+2} \right\rceil$ k-modems. For monotone orthogonal polygons, we show that every such polygon on n vertices can be illuminated with $\left\lceil \frac{n}{2k+4} \right\rceil$ k-modems and give examples which require $\left\lceil \frac{n}{2k+4} \right\rceil$ k-modems for k even and $\left\lceil \frac{n}{2k+6} \right\rceil$ for k odd.

1 Introduction

New technologies inspire new research problems, and wireless networking is a clear example of this. One such new problem is what we call here *modem illumination* of polygonal regions. Our problem arises in the following setting. It is well known that while trying to connect a laptop to a wireless modem, there are two factors that have to be considered, the *distance* to the wireless modem and, perhaps more important in most buildings, the *number of walls* separating our laptop from the wireless modem. (From now on, the term *modem* will be used to refer to a wireless modem.) We call a modem *a k-modem* if it is strong enough to transmit a stable signal through at most k walls along a straight line. Thus we say that a point p in a polygon P is covered by a k-modem m in P if the line segment joining p to m *crosses* at most k walls (edges) of P.

We point out that we allow a modem to be located at a point q *on an edge* e of P; In this case, if p is an interior point of P, the line segment connecting p to q may cross an odd number of edges of P. This follows from the fact, that the line segment connecting p to q does not *cross* the edge e of P containing q. In this paper we consider the following problem:

Modem Illumination Problem: Let P be an art gallery modeled by a polygon P with n vertices. How many k-modems located at points of P are always sufficient, and sometimes necessary, to cover all points in P?

For $k = 0$ our problem corresponds to Chvátal's Art Gallery Theorem [2] which states that $\left\lfloor \frac{n}{3} \right\rfloor$ watchmen are always sufficient and sometimes necessary to guard an art gallery with n walls. Many generalizations of the original Art Gallery problem have been studied, see [4] for a comprehensive survey.

Illumination of polygons with wireless devices has been studied recently in [3, 1] in a slightly different context, the so-called sculpture garden problem. There, each device only broadcasts a signal within a given angle of the polygon and has unbounded range. The task is to describe the polygon (distinguish it from the exterior) by a combination of the devices, meaning that for each point p in the interior of the polygon no point outside the polygon receives signals from the same devices as p. See also [5] for related problems on wireless guarding.

In this paper we provide lower and upper bounds for the Modem Illumination Problem for monotone polygons and monotone orthogonal polygons, which perfectly model most real life buildings.

For technical reasons we make the following assumptions: for a non-orthogonal monotone polygon, we assume that no two of its edges are parallel; when we speak of a polygon we refer to both the boundary and the interior of the polygon.

2 Modem Illumination of Monotone and Monotone Orthogonal Polygons

2.1 Monotone Polygons

The next lemma provides our main tool for proving upper bounds on the number of modems required to illuminate monotone polygons. It allows us to *split*

[*]Institute for Software Technology, Graz University of Technology, {oaich,thackl,bvogt}@ist.tugraz.at

[†]Instituto de Matemáticas, Universidad Nacional Autónoma de México, ruy@ciencias.unam.mx, {dflores,urrutia}@math.unam.mx

[‡]Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, Barcelona, Spain clemens@ma4.upc.edu

[§]Research supported by the Austrian FWF Joint Research Project 'Industrial Geometry' S9205-N12.

[¶]Research supported in part by MTM2006-03909 (Spain) and CONACYT of México, Proyecto SEP-2004-Co1-45876.

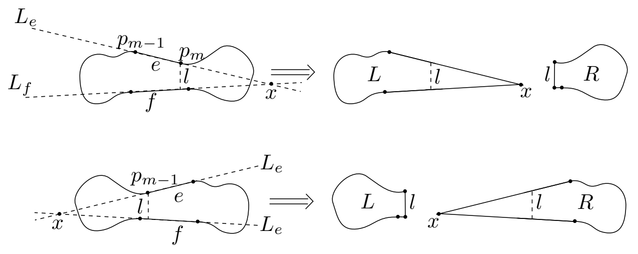


Figure 1: Illustrating the proof of Lemma 1.

monotone polygons into smaller ones, in such a way that we can illuminate these sub-polygons independently of each other.

Lemma 1 *(Splitting Lemma) Let P be a monotone polygon with vertices $p_1, p_2, \ldots, p_n$, ordered from left to right. For every positive integer $m < n$, there exist 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 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 end point of l.*
- *Denote as L' the subset of L to the left of l, and denote as R' the subset of R to the right of l; then $P = L' \cup l \cup R'$.*

Proof. Without loss of generality we assume that p_{m-1} lies on the upper polygonal chain of P. Let f be the edge of P directly below p_{m-1} and let e be the edge of P having p_{m-1} as its left endpoint. Also let $e_l = p_{m-1}$ and e_r be the left and right endpoints of e, respectively. Likewise let f_l and f_r denote the left and right endpoints of f, respectively.

We extend both e and f to straight lines, so that L_e is the straight line containing e and L_f is the straight line containing f.

Since we are assuming non-parallel edges, L_e and L_f intersect at a point x. There are two cases (see Figure 1):

1. x is to the left of p_{m-1}.
   Draw a vertical line through p_{m-1} and let l be its intersection with P. Let P^- be the subset of P to the left of l and set $L = P^- \cup l$. We define R as the polygon enclosed by:
   - the upper polygonal chain of P from e_r to p_n,
   - the lower polygonal chain of P from f_r to p_n,
   - the line segment from x to e_r and the line segment from x to f_r.
2. x is to the right of p_{m-1}.
   Draw a vertical line through p_m and let l be its intersection with P. Let P^+ be the subset of P to the right of l and set $R = P^+ \cup l$. We define L as the polygon enclosed by:
   - the upper polygonal chain of P from p_1 to p_{m-1},
   - the lower polygonal chain of P from p_1 to f_l,
   - the line segment from $e_l = p_{m-1}$ to x and the line segment from f_l to x.

Note that in both cases the three stated properties between L, R, l and P hold. □

Before stating our main theorem for monotone polygons, we remark two useful lemmas.

Lemma 2 *Every $(k+2)$-gon can be illuminated with a k-modem placed anywhere in the interior or on the boundary of the polygon.*

Proof. A $(k+2)$-gon P contains $k+2$ edges. Note that any line segment joining points of the boundary of P intersects at most k edges of P in its interior. Therefore a k-modem placed anywhere in the interior or on the boundary of P illuminates the whole polygon. □

Lemma 3 *Every $(2k+2)$-gon can be illuminated with a k-modem placed either at its $(k+2)$-th or $(k+1)$-th vertex.*

Proof. We apply Lemma 1 to P and obtain a line segment l and two monotone polygons L and R of $k+2$ vertices each; all of them satisfying the properties given by Lemma 1. We place a k-modem at an endpoint of l. By Lemma 2 this k-modem illuminates both L and R. Since in particular it also illuminates L' and R', by Lemma 1 all of P is illuminated. □

We remark that for the particular cases of $k = 1, 2, 3$, Lemma 3 can be strengthened as follows (for lack of space, we omit the proofs):

Lemma 4 *For every monotone heptagon there exists a point inside the polygon, between its second and sixth vertex, where a 1-modem can be placed to illuminate the whole polygon.*

Proof. Omitted. □

Lemma 5 *Every monotone 8-gon can be illuminated with one 2-modem, placed either at its fourth or fifth vertex.*

Proof. Omitted. □

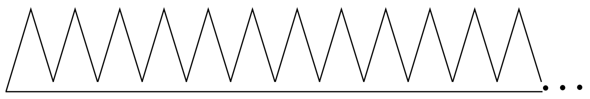


Figure 2: A monotone n-gon requiring $\lceil \frac{n}{2k+2} \rceil$ k-modems.

Lemma 6 *Every monotone* 9*-gon can be illuminated with one* 3*-modem placed at its fifth vertex.*

Proof. Omitted. □

We are now ready to state our main theorem.

Theorem 7 *Every monotone n-gon can be illuminated with $\lceil \frac{n}{2k} \rceil$ k-modems, and there exist monotone n-gons that require at least $\lceil \frac{n}{2k+2} \rceil$ k-modems to be illuminated.*

Proof. An example achieving the lower bound is given in Figure 2. So it remains to prove the upper bound. Using Lemma 1 recursively, we split the n-gon into $m = \lceil \frac{n}{2k} \rceil$ $(2k+2)$-gons as follows: apply Lemma 1 to P and obtain a line segment l_1, a monotone polygon L_1 of $2k+2$ vertices and a monotone polygon R_1 of $n-2k$ vertices; all satisfying the properties of Lemma 1. Apply now Lemma 1 to R_1 to obtain a line segment l_2, a monotone polygon L_2 of $2k+2$ vertices and monotone polygon R_2 of $n-4k$ vertices.

Continue this process and obtain the $\lceil \frac{n}{2k} \rceil$ monotone $(2k+2)$-gons $L_1, L_2, \ldots, L_m$ and the line segments $l_1, \ldots, l_{m-1}$; all satisfying the properties of Lemma 1.

For each L_i $(1 < i < m)$, let Q_i be the subset of L_i to the left of l_i and to the right of l_{i-1}. For L_1 and L_m, let Q_1 be the subset of L_1 to the left of l_1 and Q_m the subset of L_m to the right of l_{m-1}.

By Lemma 3, each L_i can be illuminated with a k-modem placed in Q_i. Note that since $P = (\bigcup Q_i) \cup (\bigcup l_i)$, this $\lceil \frac{n}{2k} \rceil$ modems of power k illuminate all of P. □

The proof of the upper bound in Theorem 7 uses Lemma 3, however for $k = 1, 2, 3$ the corresponding strengthened version of Lemma 3 can be used instead to obtain the following (better) upper bound:

Theorem 8 *For $k = 1, 2, 3$, any monotone n-gon can be illuminated using $\left\lceil \frac{n}{k+4} \right\rceil$ k-modems.*

Proof. Omitted. □

We remark that for the particular case of 1-modems, Theorem 8 gives an upper bound of $\lceil \frac{n}{5} \rceil$ 1-modems, and that there exist monotone n-gons (see Figure 3) that achieve this as a lower bound.

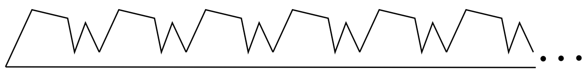


Figure 3: A monotone n-gon requiring $\lceil \frac{n}{5} \rceil$ 1-modems.

2.2 Monotone Orthogonal Polygons

In this section we give lower and upper bounds on the number of k-modems needed to illuminate orthogonal and monotone orthogonal polygons. Recall that for the main motivation of our research, namely to place modems inside buildings in order to cover the interior of the building with wireless reception, often orthogonal polygons are a quite realistic scenario.

Proposition 9 *Every orthogonal at most $(k+4)$-gon P is illuminated by a k-modem placed anywhere in the interior or on the boundary of P.*

Proof. Any line segment l with endpoints inside or on the boundary of the polygon P cannot properly (i.e., in the interior of l) intersect any of the leftmost vertical, topmost horizontal, rightmost vertical or bottom-most horizontal edge of P. Therefore, being at most $k+4$ edges in total, at most $k+4-4=k$ edges are intersected by l. Thus, a k-modem placed anywhere inside or on the boundary of P illuminates the whole polygon. □

Proposition 10 *For any x-monotone orthogonal $(k+5)$-gon there is a point on its leftmost (or rightmost) edge where a k-modem can be placed to illuminate the polygon.*

Proof. If (at least) one of the two horizontal edges adjacent to the leftmost vertical edge, say e, is not the topmost or bottom-most horizontal edge, respectively, then placing the modem at the common vertex of e and the leftmost vertical edge illuminates the polygon. This follows from the proof of Proposition 9 and the fact that e does not block the rays of the modem.

Otherwise consider the two horizontal edges adjacent to the rightmost vertical edge. At least one of them is not the topmost or bottom-most horizontal edge. Let e' be this edge and w.l.o.g. assume that the interior of the polygon lies below e'. The horizontal line supporting e' intersects the leftmost vertical edge in its interior. Placing the modem on the leftmost vertical edge and below this intersection point illuminates the whole polygon by arguments similar to the previous case. Note that this is where we need the x-monotonicity of the polygon to guarantee that e' cannot block the rays of the modem. □

Using the previous observations we now can prove the following.

Proposition 11 *Every x-monotone orthogonal $(2k+6)$-gon can be illuminated with a k-modem.*

Proof. If k is even, we split the polygon vertically into two $(k+4)$-gons and place a k-modem in their common intersection; Proposition 9 ensures that the whole polygon is illuminated.

For odd k, split the polygon into one $(k+3)$-gon and one $(k+5)$-gon. By Proposition 10, there exists a point in their common intersection where a k-modem can be placed to illuminate the $(k+5)$-gon; Proposition 9 ensures that the $(k+3)$-gon is also illuminated. □

Our main result for monotone orthogonal polygons is thus:

Theorem 12 *Every x-monotone orthogonal polygon on n vertices can be illuminated with $\lceil \frac{n-2}{2k+4} \rceil$ k-modems.*

Proof. Split the n-gon into $(2k+6)$-gons and apply Proposition 11. □

For the case when k is even, the bound of Theorem 12 is tight, as shown by Figure 4 (right), where a monotone orthogonal n-gon requiring $\lceil \frac{n-2}{2k+4} \rceil$ k-modems to be illuminated is shown. For odd k, the same example gives a lower bound of $\lceil \frac{n-2}{2k+6} \rceil$.

However, for 1-modems, an example of a monotone orthogonal n-gon requiring $\lceil \frac{n-2}{6} \rceil$ 1-modems to be illuminated is shown in Figure 4 (left). Thus in this case the bound is also tight.

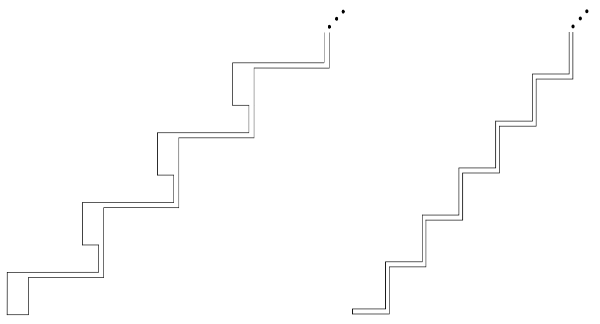


Figure 4: Lower bound constructions for monotone orthogonal polygons.

3 Conclusions

Inspired by current wireless networks, we studied a new variant of the classic polygon-illumination problem. To model the way wireless devices communicate within a building, we now allow light to cross a variable number of walls.

Using as a main tool the Splitting Lemma that allows us to divide a polygon into simpler, overlapping polygons, we give an upper bound on the number of k-modems needed to illuminate any given monotone polygon. We also presented a family of monotone polygons that need at least a number of k-modems close to that of our upper bound.

Moreover we studied the particular case when the monotone polygons are orthogonal, where we have been able to give, in most cases, tight bounds.

The natural open problem remaining is to close the gap between lower and upper bounds for both types of polygons, monotone and monotone orthogonal.

The modem illumination problem for general polygons, has proved to be rather challenging. We believe that obtaining tight bounds for this case is a non-trivial problem. At the moment the best lower bounds we have, are those we have for monotone polygons, and no significant upper bounds are known to us. Noteworthy, we might not be surprised if the lower bounds for general polygons happen to be realized by monotone polygons also; note that this happens for the $\lfloor n/3 \rfloor$ lower bound for the classical polygon illumination problem.

4 Acknowledgments

We would like to thank Wolfgang Aigner, Franz Aurenhammer and Bernhard Kornberger, for helpful discussions.

References

[1] D. Christ, M. Hoffmann, Y. Okamoto, T. Uno, Improved Bounds for Wireless Localization. *Proc. 11th Scandinavian Workshop on Algorithm Theory*, pp. 77–89, 2008.

[2] V. Chvátal, A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18:39–41, 1975.

[3] D. Eppstein, M.T. Goodrich, N. Sitchinava, Guard Placement for Efficient Point-in-Polygon Proofs. *Proc. 23rd Symposium on Computational Geometry*, pp. 27–36, 2007.

[4] J. Urrutia, Art Gallery and Illumination Problems. *In: J.R. Sack, J. Urrutia (eds.) Handbook of Computational Geometry*, pp. 973–1027, Elsevier Science Publishers B.V., 2000.

[5] Y. Wang, C. Hu, Y. Tseng, Efficient Placement and Dispatch of Sensors in a Wireless Sensor Network. *IEEE Transactions on Mobile Computing*, 7:262–274, 2008.

Low-Cost Tours for Nearsighted Watchmen with Discrete Vision

Sándor P. Fekete* Christiane Schmidt* †

Abstract

Computing an optimal watchman tour, a minimum guard cover, or a minimum length milling tour (i.e., a closed path for a tool such that every point in the polygon is touched by the tool) are natural and classical geometric optimization problems. We consider a generalization of all three problems, motivated by applications from robotics: Given a (simple) polygon and a visibility range, find a closed path P and a set of scan points $\mathrm{S} \subset P$, such that (i) every point of the polygon is within visibility range of a scan point; and (ii) path length plus weighted sum of scan number along the tour is minimized.

We demonstrate that this problem is NP-hard, even for the special cases of rectilinear polygons and L_∞ scan range 1, and negligible small travel cost or negligible travel cost. Our main result is a 2.5-approximation for any linear combination of travel cost and scan cost.

1 Introduction

In recent years, the development of real-world autonomous robots has progressed to the point where actual visibility-based guarding, searching, and exploring become very serious practical challenges, offering new perspectives for the application of algorithmic solutions. However, some of the technical constraints that are present in real life have been ignored in theory; taking them into account gives rise to new algorithmic challenges, necessitating further research on the theoretical side, and also triggering closer interaction between theory and practice. Two of these constraints are (a) limited visibility range; and (b) requirement to stop when scanning the environment. These constraints give rise to the *Myopic Watchman Problem with Discrete Vision* (MWPDV), which is the subject of this paper.

Related Work. Closely related to practical problems of searching with an autonomous robot is the classical theoretical problem of finding a *shortest watchman tour*; e.g., see [5, 6]. Planning an optimal set of scan points (with unlimited visibility) is the *art gallery problem* [12]. Finally, visiting all grid points of a given set is a special case of the classical *Traveling Salesman Problem* (TSP), see [10]. Two generalizations of the TSP are the so-called *lawnmowing* and *milling problems*: Given a cutter of a certain shape, e.g., an axis-aligned square, the *milling problem* asks for a shortest tour along which the (center of the) cutter moves, such that the entire region is covered and the cutter stays inside the region all the time. Clearly, this takes care of the constraint of limited visibility, but it fails to account for discrete visibility. At this point, the best known approximation method for milling yields a 2.5-approximation [2]. Other results for TSP with neighborhoods include [7]; further variations arise from considering online scenarios, either with limited vision [4] or with discrete vision [8, 9], but not both. Finally, [1] consider covering a set of points by a number of scans, and touring all scan points, with objective cost being a linear combination of scan cost and travel cost; however, the set to be scanned is discrete, and scan cost is a function of the scan radius, which may be small or large.

Our Results. We point our that even extreme cases of the MWPDV are NP-hard. Our main result is a 2.5-approximation for the case of grid polygons, based on the 2.5-approximation by Arkin, Fekete, and Mitchell [2].

2 Notation and Preliminaries

We are given a rectilinear polygon P with integer coordinates, i.e., a polygon that can be described as the union of axis-parallel unit grid squares (*pixels*); these polygons are called *polynominoes*. Our robot, R, has discrete vision, i.e., it can perceive its environment when it stops at a point and performs a scan, which takes c time units. Moreover, the visibility range is limited: The points where R stops are called scan points; from a scan point p, only the unit sphere in L_∞-norm around p is visible to R, i.e., the four pixels incident to p. A set S of scan points *covers* the polygon P, iff every point in P is visible from and within visibility range of a point $p \in \mathrm{S}$.

We search for a tour T and a set of scan points S that covers P, such that the total travel and scan time is optimal, i.e.,

$$t(T) = c \cdot |\mathcal{S}| + L(T) \qquad (1)$$

is minimized. The following lemma allows us to focus on visiting and scanning at grid points.

*Algorithms Group, Braunschweig Institute of Technology, Germany. Email: {s.fekete,c.schmidt}@tu-bs.de, http://www.ibr.cs.tu-bs.de/alg

†Supported by DFG Focus Program "Algorithm Engineering" (SPP 1307) project "RoboRithmics" (Fe 407/14-1).

Lemma 1 *For a rectilinear pixel polygon P there exists an optimum myopic watchman tour T^* such that all scan points are located on grid points:*

$$\exists T^* \forall (x_s, y_s) \in \mathcal{S}(T^*) : x_s \in \mathbb{Z} \wedge y_s \in \mathbb{Z}. \qquad (2)$$

Proof. Let T be an optimal tour, with scan points not located on grid points. Consider the vertical and horizontal strips of pixels in P of maximal length. W.l.o.g., we start with the horizontal strips. For every strip, we shift the scans, such that the x-coordinates are integers (starting from the boundary, i.e., with distance 1 to the boundary if possible, and away from non-reflex corners). The tour will not be longer, we cover not less; in case we are able to reduce the number of scans per strip by one we have a contradiction to t being optimal. After applying this to all horizontal stripes we proceed analogously for the vertical strips. Hence, we have an optimal tour, with all scan points located on grid points. □

3 NP-Hardness

We point out that even extreme cases of the MWPDV are NP-hard:

Theorem 2 *(1) The MWPDV is NP-hard, even for polynominoes and small or no scan cost, i.e., $c \ll 1$ or $c = 0$.*

(2) The MWPDV is NP-hard, even for polynominoes and small travel cost, i.e., $c \gg 1$.

(3) The MWPDV is NP-hard, even for polynominoes and no travel cost.

Proof Sketch. The first claim is a result of the hardness of minimum cost milling, see [2]. The second claim is an easy consequence of the NP-hardness of `Hamiltonicity of Grid Graphs` (HGG) [10]: Given an instance G of HGG with n vertices, turn it into an instance of MWPDV by scaling the grid graph G by a factor of two, and replacing each grid point of G by a 2x2-square. This yields a canonical set of n scan points that is contained in any optimal WMPDV tour; traveling these with a tour length $2n$ is possible if and only the graph G is Hamiltonian.

The third claim is closely related to a minimum cover problem by visibility discs; however, the MWPDV requires that the scan points must be inside of the polygonal region. We give a proof sketch along the lines of [3], based on a reduction of the NP-hard problem `Planar 3SAT`, a special case of 3SAT in which the variable-clause incidence graph H is planar. As a first step, we construct an appropriate planar layout of the graph H, e.g., by using the method of Rosenstiehl and Tarjan [13]. This layout is turned into a grid polygon by representing the variables, the clauses and the edges of G. An example for the variable component is given in Figure 1.

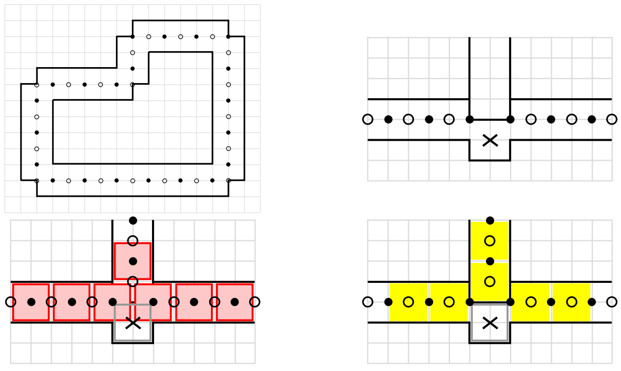


Figure 1: Top row: The polygonal piece for a variable and the connection of an edge corridor. Bottom row: a placement corresponding to "true" (left), and a placement corresponding to "false" (right). The light gray scan is used in both cases.

The variable gadgets allow two ways for locating a minimum number of scan points. The first (the black points in Figure 1) relates to a setting of "true", the other (the circles in Figure 1) to a setting of "false". For the variable setting "true" the scan squares are pushed further into the edge corridor. These edge corridors are similar to the variable-circles—bendings are done accordingly. In order to have the same number of points and circles, edge corridors may be added, that do not end in another polygonal piece, but assure this parity (with circles at the edge corridor).

A clause component is given in Figure 2. Edge corridors of the three associated variables (each with the appropriate truth setting) meet in the polygonal piece for the clause (dark gray in Figure 2). If and only if the clause is satisfied, i.e., if at least in one edge corridor a scan square is placed at a black point, three additional scans suffice to cover this polygonal piece. Otherwise, four scans are necessary.

Given the components defined above we can compute the parameter k, the number of scan squares necessary to cover the entire resulting polygon P. k is polynomial in the number of vertices of G and part of the input. All vertices of the resulting P have integer coordinates of small size, their number is polynomial in the number of vertices of G. This shows that the problem is NP-hard.

4 Approximation

Our approximation proceeds in two steps:

(I) Construct a set of scan points that is not larger than 2.5 times a minimum cardinality scan set.

(II) Construct a tour that contains all constructed scan points and that does not exceed 2.5 times the cost of an optimum milling tour.

The main idea for the second step is based on the $O(n \log n)$-time 2.5-approximation algorithm for

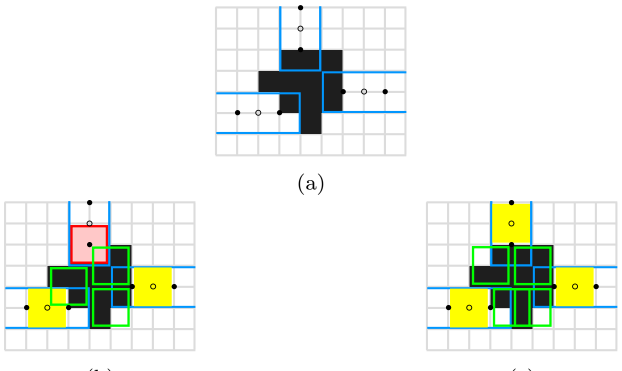


Figure 2: A clause component with the polygonal piece (dark gray) and the three according edge corridors (blue)(a). A placement corresponding to "true" (b), and a placement corresponding to "false" (c). The green squares are the scans necessary inside the clause.

milling from Arkin et al. [2]. The resulting tour consists of three parts, see Figure 3 for an example, L_{OPT} being the optimal milling tour length:

(1) a "boundary" part: $B \subset P$ is the inward offset region of all points within P that are feasible placements for the center of the milling cutter. For a milling problem, B is connected. Tracing the boundary δB of B, let $P_{\delta B}$ denote the region milled by this route. (δB may not be connected (if P features holes), the pieces are δB_i.) The length of δB, $L_{\delta B}$ is a lower bound on L_{OPT}.

(2) a "strip" part: $P_{int} := P \backslash P_{\delta B}$—if nonempty—can be covered by a set of k horizontal strips Σ_i. The y-coordinates of two strips differ by multiples of 2. Then, let $L_{str} = \sum_{i=1}^{k} L_{\Sigma_i}$ and this is again a lower bound on the length of an optimal milling tour: $L_{OPT} \geq L_{str}$.

(3) a "matching" part: the strips and the boundary tour have to be combined for a tour. For that purpose, consider the endpoints of strips on δB_i: every δB_i contains an even number of such endpoints. Hence, every δB_i is partitioned into two disjoint portions, $M_1(\delta B_i)$ and $M_2(\delta B_i)$. Using the shorter of these two ($M_*(\delta B_i)$) for every δB_i we obtain for the combined length, L_M: $L_M \leq L_{str}/2 \leq L_{OPT}/2$.

The graph with endpoints of strip lines plus the points where strip line touches a δB_i as vertices is connected by three types of edges—the center lines of the strips, the δB_is and the $M_*(\delta B_i)$s. Every vertex has degree 4, hence, an Eulerian tour gives a feasible solution.

Now we describe how to construct a covering set of scan points:

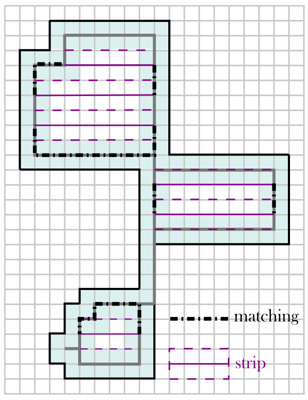


Figure 3: A polygon P, consisting of pixels. The parts for the approximative milling tour are indicated: $P_{\delta B_i}$ in light blue, δB_i in gray.

1. Let S_{4e} be the "even quadruple" centers of all 2x2-squares that are fully contained in P, and which have two even coordinates.

2. Remove all 2x2-squares corresponding to S_{4e} from P; in the remaining polyomino P_{4e}, greedily pick a maximum disjoint set S_{4o} of "odd quadruple" 2x2-squares.

3. Remove all 2x2-squares corresponding to S_{4o} from P_{4e}; greedily pick a maximum disjoint set S_3 of "triple" 2x2-squares that cover 3 pixels each in the remaining polyomino $P_{4e,4o}$,

4. Remove all 2x2-squares corresponding to S_3 from $P_{4e,4o}$; in the remaining set $P_{4e,4o,3}$ of pixels, no three can be covered by the same scan. Considering edges between pixels that can be covered by the same scan, pick a minimum set of ("double" S_2 and "single" S_1) scans by computing a maximum matching.

Claim 1. The total number of scans is at most 2.5 times the size of a minimum cardinality scan set.

Claim 2. All scan points lie on a 2.5-approximative milling tour.

For the first claim, let $s_{\min}$ be the size of a cardinality scan set. Observe that $S_{4e} \cup S_{4o}$ corresponds to a set of disjoint 2x2-squares that are fully contained in P; hence, it follows from a simple area argument that $|S_{4e} \cup S_{4o}| \leq s_{\min}$.

When considering the set of pixels in $P_{4e,4o}$, not more than three can be covered by the same scan; in $P_{4e,4o,3}$ we compute an optimal solution. This implies that the only way to get a smaller cover in $P_{4e,4o}$ is to change the choice of triple scans; this means that the pixels covered by a triple scan have to be allocated differently. Taking into account that $P_{4e,4o}$ does not contain any 2x2-squares, a simple case analysis shows that the only possible improvement replaces a triple, a double and a single by two triples (as shown in the left part of Figure 4). With $s^4_{\min}$ being the minimum number of scans required for covering $P_{4e,4o}$, we get $|S_3 \cup S_2 \cup S_1| \leq \frac{3}{2}s^4_{\min} \leq \frac{3}{2}s_{\min}$. In total we get a solution with not more than $\frac{3}{2}s_{\min}$ scans.

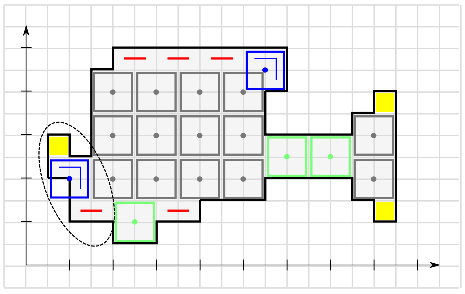


Figure 4: An example for our approximation method: The set of "even quadruple"scans is shown in grey; the "odd quadruple" scans are green. A possible (greedy!) set of "triple" scan is shown in blue, leaving the maximum matching (and the corresponding "double scans") shown in red. The leftover single pixels are yellow. The ellipse indicates a part that is covered by three scans instead of two: the triple scan with adjacent single and double scans could be covered by two triple scans.

For the second claim, we use a milling tour constructed as in [2] and described above: Choosing the strips to be centered on even y-coordinates allows us to visit all scan points in S_{4e}. Clearly, all pixels in P_{4e} are adjacent to the boundary of other scans involve boundary pixels, allowing them to be visited along the "boundary" part. (One minor technical detail is shown in Figure 3: In order to visit the center of a triple scan, we need to reroute the boundary part of the tour to run through a reflex vertex; this does not change the tour length.)

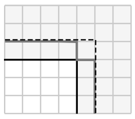


Figure 5: Modifying the boundary part of $W(t)$: at a reflex vertex, the gray path is used instead of δB_i.

This concludes the proof. We summarize:

Theorem 3 *A polyomino P allows a MWPDV solution that contains at most 2.5 times the minimum number of scans necessary to scan the polygon, and has tour length at most 2.5 times the length of an optimum milling tour.*

5 Conclusion

We have introduced the NP-hard Myopic Watchman Problem with Discrete Vision. For the case of L_∞ range in rectilinear polygons, we gave a 2.5-approximation for both tour length and number of scans. Open questions include the following:

- Can we give an approximation in the case of L_2 visibility range, i.e., for circles instead of squares?
- Can we improve the approximation factor?

For large visibility range and large scan cost, the problem turns into minimum guard cover, for which no constant-factor approximation is known; however, we hope to achieve positive results for polygons that are "fat" in terms of scan range vs. feature size. Improving the approximation factor requires improving the factor for milling; there is some indication that there may even be a PTAS for the latter [11], so we are somewhat optimistic.

References

[1] H. Alt, E. M. Arkin, H. Brönnimann, J. Erickson, S. P. Fekete, C. Knauer, J. Lenchner, J. S. B. Mitchell, and K. Whittlesey. Minimum-cost coverage of point sets by disks. In *Symposium on Computational Geometry*, pages 449–458, 2006.

[2] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Comput. Geom. Theory Appl.*, 17(1-2):25–50, 2000.

[3] C. Baur and S. P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.

[4] A. Bhattacharya, S. K. Ghosh, and S. Sarkar. Exploring an unknown polygonal environment with bounded visibility. In *International Conference on Computational Science (1)*, volume 2073 of *LNCS*, pages 640–648. Springer, 2001.

[5] W.-P. Chin and S. Ntafos. Optimum watchman routes. *Proc. 2nd ACM Symposium on Computational Geometry*, 28(1):39–44, 1988.

[6] W.-P. Chin and S. C. Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6:9–31, 1991.

[7] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003.

[8] S. P. Fekete and C. Schmidt. Polygon exploration with discrete vision. In *23rd European Workshop Comput. Geom.*, pages 86–89. Universität Graz, 2007.

[9] S. P. Fekete and C. Schmidt. Polygon exploration with discrete vision. *CoRR*, abs/0807.2358, 2008.

[10] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[11] J. S. B. Mitchell. Personal communication, 2009.

[12] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Internat. Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[13] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry*, 1:343–353, 1986.

Natural Wireless Localization is NP-hard

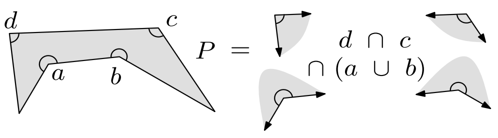
Tobias Christ[1] Michael Hoffmann[1] Yoshio Okamoto[2]

[1] Institute for Theoretical Computer Science, ETH Zürich, {christt,hoffmann}@inf.ethz.ch
[2] Tokyo Institute of Technology, okamoto@is.titech.ac.jp

Abstract

We consider a special class of art gallery problems inspired by wireless localization. Given a polygonal region $\mathcal{P}$, place and orient guards each of which broadcasts a unique key within a fixed angular range. In contrast to the classical art gallery setting, broadcasts are not blocked by the boundary of $\mathcal{P}$. At any point in the plane one must be able to tell whether or not one is located inside $\mathcal{P}$ only by looking at the set of keys received. We prove NP-hardness of one variant of the problem, namely the *natural* setting where guards may be placed aligned to a boundary edge or two consecutive boundary edges of $\mathcal{P}$ only.

Introduction. Art gallery problems are a classic topic in discrete and computational geometry. A new direction has recently been introduced by Eppstein, Goodrich, and Sitchinava [1]. They propose to modify the concept of visibility by not considering the edges of the polygon/gallery as blocking. This changes the problem quite drastically because it breaks up a certain locality where the shape of the polygon dictates the possible placement of guards. A basic ingredient in hardness proofs for the classical setting is a small pocket/spike of the polygon which can only be guarded from a nearby point because the bounding polygon edges shield it away from the rest of the world. This argument breaks down if the edges do not block visibility. The motivation for this model stems from communication in wireless networks where the signals are not blocked by walls, either. For illustration, suppose you run a café (modeled, say, as a simple polygon P) and you want to provide wireless Internet access to your customers. But you do not want the whole neighborhood to use your infrastructure. Instead, Internet access should be limited to those people who are located within the café. To achieve this, you can install a certain number of devices, let us call them guards, each of which broadcasts a unique (secret) key in an arbitrary but fixed angular range. The goal is to place guards and adjust their angles in such a way that everybody who is inside the café can prove this fact just by naming the keys received and nobody who is outside the café can provide such a proof. Formally this means that P can be described by a monotone Boolean formula over the keys, that is, a formula using the operators And and Or only, negation is not allowed. (See [1, 2, 3].)



Notation. A *guard* g is a closed subset of the plane, whose boundary ∂g is described by a vertex v and two rays emanating from v. The ray that has the interior of the guard to its right is called the *left ray* ℓ_g, the other one is called the *right ray* r_g. The *angle* of a guard is the interior angle formed by its bounding rays. For a guard with angle π, the vertex is not unique. A *guarding* $\mathcal{G}(\mathcal{P})$ for a polygonal region $\mathcal{P}$ is a set of guards such that there is a formula composed of this set and the operators union and intersection that defines $\mathcal{P}$. Natural locations for guards are the vertices and edges of the polygonal region. A guard which is placed at a vertex of $\mathcal{P}$ is called a *vertex guard*. A vertex guard is *natural* if it covers exactly the interior angle of its vertex [1]. A guard placed anywhere on the line given by an edge of $\mathcal{P}$ and broadcasting within an angle of π to the inner side of the edge is called a *natural edge guard*. A *natural guarding* is a guarding consisting of natural vertex and natural edge guards only [3]. From now on we consider natural guardings only.

The Natural Wireless Localization Problem. Given a finite union of simple polygons $\mathcal{P}$ and an integer k, is there a natural guarding for $\mathcal{P}$ using k guards?

Theorem 1 *The Natural Wireless Localization Problem is NP-complete.*

The problem is in NP. Given a set of natural guards we can check in polynomial time whether they can describe $\mathcal{P}$. To show the NP-completeness we reduce the Vertex Cover Problem to the Wireless Localization Problem. It is sufficient to consider 3-connected 3-regular planar graphs (see [4]). Let $G = (V, E)$ be such a graph of order n. Embed the vertices V into the plane, that is, think of G as a geometric graph. Denote the line defined by an edge $e \in E$ by $\overline{e}$. The set of edges incident to a vertex $v \in V$ is denoted by $E(v)$. Consider the line arrangement $L = \{ \overline{e} \mid e \in E\}$ defined by G. Add all perpendicular bisectors of edges, resulting in an *extended line arrangement* L'. We want L' to be in general position in the following *weak sense*: only the three lines defined by the edges incident to a vertex v meet in v and only $\overline{e}$ and its perpendicular bisector meet in the midpoint of e denoted by $M(e)$. Furthermore, we want the embedding to be strictly convex.

Lemma 2 *Given a 3-regular and 3-connected planar graph $G = (V, E)$ it is possible to embed V into an $O(n^5) \times O(n^5)$ grid in time polynomial in n such that the extended line arrangement L' of G is in general position in the weak sense and every face is strictly convex.*

Proof. A drawing of G on an $O(n^2) \times O(n^2)$ grid in which all faces are strictly convex polygons can be obtained in linear time [5]. Let G be such a geometric graph on a $K \times K$ grid with $K \in O(n^2)$. The vertices and the midpoints of edges are called *points of interest*. Perturb the vertices locally one by one to finally get a new drawing which is in general position in the weak sense, that is, no lines of L' intersect the points of interest except those who have to. (We still denote the drawing after perturbation by G and the extended line arrangement by L'.)

An easy calculation reveals that on a $K \times K$ square grid the distance between any grid point v and any line l disjoint from v that is defined by two grid points is at least $1/(\sqrt{2}K)$. Define $\varepsilon := 1/(2\sqrt{2}K)$. If we perturb the vertices by less than ε, then the drawing remains strictly convex. (Destroying a convex face corresponds to moving a vertex v of the face and a line l defined by two other vertices of the face such that after the perturbation v lies on the other side of l. The distance of v and l is at least 2ε, therefore after having perturbed every vertex by less than ε, v cannot have changed sides.) In the same way one might argue that the drawing remains crossing-free, but this is not essential for our purposes.

Consider a small $k \times k$ grid around every vertex v inside an ε-ball around v and choose a new location for v in this grid as follows: Assume that we already have perturbed the first m vertices and that the extended line arrangement of the subgraph induced by the first m vertices is in general position in our weak sense. Perturbing the next vertex v we want to ensure that neither v or one of the midpoints of its incident edges come to lie on a line of L' nor it induces a new line in L' that intersects a point of interest.

The extended line arrangement L' is built on $3n$ lines. (There are $3n/2$ edges giving rise to two lines each.) Therefore, we have $n + 3n/2$ points of interest. It is forbidden to put v

- on any of the lines from L', except for those lines $\overline{e}$ that are defined by an edge $e \in E(v)$ (less than $3n$ forbidden lines),
- on any line such that placing v there would result in one of the midpoints $M(e)$ of an edge $e \in E(v)$ lying on a forbidden line (less than $3 \cdot 3n = 9n$ forbidden lines),
- on any line defined by one of the neighbors of v and an arbitrary other point of interest (less than $(15/2)n$) forbidden lines),
- on any circle C centered at a point q of interest and passing through one of the neighbors of v (less than $(15/2)n$ forbidden circles), as placing v on C would cause the perpendicular bisector of the corresponding edge to pass through q.

All of these less than $27n$ forbidden loci are either lines or circles, each of which hits at most k of the k^2 grid points of the small grid around v. Choosing k larger than $27n$ we can always be sure to find an appropriate placement for v. Recall that this small grid is placed within an ε-ball around v. As $k = O(n)$ and $\varepsilon = \Omega(n^{-2})$, the distance between any two distinct points on a small grid is $\Omega(n^{-3})$. Overall, we refine the $K \times K$ grid by a factor of $O(n^3)$, which results in an $O(n^5) \times O(n^5)$ grid. $\square$

Let δ denote the minimum distance between any point q of interest and a line of L' disjoint from q. After perturbation the vertices lie on an $O(n^5) \times O(n^5)$ grid. Refining it by a factor of 2 ensures that all midpoints of edges are grid points, too. As on a $K \times K$ grid any grid point has distance at least $1/(\sqrt{2}K)$ to any disjoint line defined by two grid points, $\delta = \Omega(n^{-5})$.

The main steps of the reduction are the following. We successively replace every vertex by a small convex hexagon. There are exactly two ways to guard such a polygon optimally: if the vertices are numbered in order, either put guards on all even vertices or put guards on all odd vertices. The idea is that one way of guarding this vertex gadget corresponds to the vertex being in the vertex cover and the other way of guarding it corresponds to the vertex not being in the cover.

Then, successively replace every edge by an edge gadget, that is, a simple polygon having the property that it needs at least 4 natural guards unless there is a ray of some other guard passing through the gadget, in which case it needs 3 guards only. Thus, the edge gadget "asks" one of its incident vertex gadgets to help out with such a ray, which corresponds to the edge asking for one of its incident vertices to be in the cover.

Finally, we will add a "punishment gadget" to every vertex, to ensure that every vertex put into the cover is paid for.

Vertex gadget. Every vertex $v \in V$ is replaced by a convex hexagon centered at v. Let $E(v) = \{e_1, e_2, e_3\}$ be the edges incident to v. Let $P(v)$ be a polygon that has two edges collinear to e_1, two edges collinear to e_2 and two edges collinear to e_3, all of them having length λ. We can avoid creating edges that are on the same line as edges already defined in other vertex gadgets by varying the edge length $0 < \lambda < \delta$. Observe that every line defined by the edges of $P(v)$ is parallel to one of the lines in the extended line arrangement L' with distance at most δ.

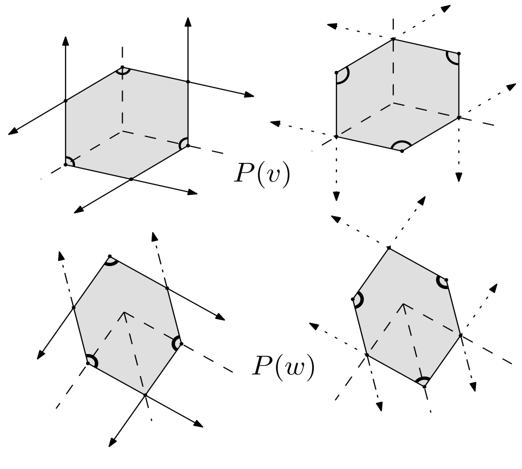


Figure 1: The two ways to guard $P(v)$ for a vertex v and $P(w)$ for a pointed vertex w.

Observation 1 *There are exactly two ways to guard the vertex gadget $P(v)$ of an vertex v using 3 guards only: Either the rays of the guards go into the same direction as the edges of v, which we call the* first way *to guard it, or they go into the other direction, which we call the* second way.

In the special case of a *pointed* vertex, that is, a vertex for which all incident edges lie within an angle of less than π, the rays parallel to the middle edge go into the other direction. See Figure 1.

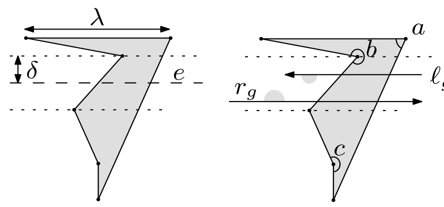


Figure 2: An edge gadget $Q(e)$.

Edge gadget. For every edge $e \in E$ we insert a polygon $Q(e)$ such that no line through any edge defined in some other gadget intersects $Q(e)$, except for the lines parallel to e that are defined by the two vertex gadgets corresponding to the endpoints of e. Furthermore, we have to ensure that no line defined by an edge of $Q(e)$ intersects any other edge gadget already created. If λ is chosen to be sufficiently small, the lines defined by $Q(e)$ remain inside a δ-strip around the perpendicular bisector of e within our area of interest (up to outermost intersection of the perpendicular bisector with other lines of the extended line arrangement). Hence the lines defined by $Q(e)$ do not intersect any other edge gadget, as these gadgets are located at an intersection of other lines of L'. Note that choosing a very small λ and therefore a "very flat" $Q(e)$ does not change the properties of $Q(e)$ as far as guardings are concerned.

For an edge e that is the middle edge of a pointed vertex on the outer face, do not place $Q(e)$ on the midpoint $M(e)$ but somewhere within the outer face along $\overline{e}$. Note that at this point we use that G forms a strictly convex embedding: As all pointed vertices lie on the outer face, two pointed vertices can only be connected by an edge of the outer face, which cannot be a middle edge. In other words, at most one endpoint of e can be pointed.

If $Q(e)$ is hit by a ray r_g of a guard g or a left ray $\ell_{g'}$ of a guard g' from the incident vertex gadgets, then there $Q(e)$ can be guarded using 3 guards only. (See Figure 2, $Q(e) = a \cap b \cap (c \cup g)$ or $Q(e) = a \cap b \cap (c \cup g')$, respectively.) If there are no such rays crossing $Q(e)$, then at least 4 guards are needed to guard $Q(e)$, see Figure 3. (If there was a guarding using 3 guards only, then there would have to be a natural vertex guard on every second vertex. In either case there would be a pair of points (depicted by crosses in Figure 3), one inside $Q(e)$ the other outside of $Q(e)$, that could not be distinguished by the guards.)

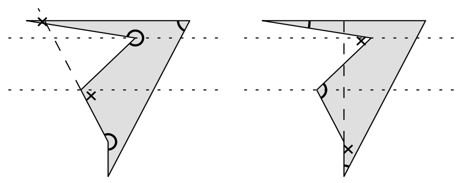


Figure 3: An edge gadget hit by no ray.

Observation 2 *At least 4 guards are needed for $Q(e)$ unless one of the incident vertex gadgets is guarded in the first way, in which case 3 guards suffice.*

The punishment gadget. For every vertex v we add a punishment gadget similar to the edge gadgets. Choose an edge $e \in E(v)$ and place a polygon $R(v)$ as shown in Figure 2 somewhere along $\overline{e}$, (seen from v) on the side opposite to where the edge gadget is. (For a pointed vertex, pick one of the two edges incident to the reflex angle.) Again make sure not to create any kind of interference with other gadgets. We can always find an appropriate placement for the gadgets by putting them somewhere far away from everything else and choosing λ small enough. For lack of space, we have to defer details to the full paper.

Observation 3 *Every punishment gadget needs at least 4 guards if there are no rays of other guards intersecting it. If its corresponding vertex gadget is guarded in the second way, then there is a guarding using only 3 guards.*

All in all we get a finite union of simple polygons $\mathcal{P}(G)$ composed of all vertex gadgets, all edge gadgets and all punishment gadgets. It is possible to construct $\mathcal{P}(G)$ in polynomial time. Let $n = |V|$ be the number of vertices and $m = |E| = 3n/2$ the number of edges. The following lemma completes the proof of Theorem 1.

Lemma 3 *There is a vertex cover of G using k vertices if and only if there is a natural guarding for $\mathcal{P}(G)$ using at most $3m + 6n + k$ guards.*

Proof. If there is a vertex cover of G of size k we can find a guarding of $\mathcal{P}(G)$ as follows. Guard every vertex that is in the cover in the first way, every vertex that is not in the cover in the second way. For every vertex guarded the first way we have to put 4 guards on the corresponding punishment gadget, for every vertex we covered the second way only 3. For all edge gadgets 3 guards are sufficient, because since we started from a vertex cover there will be the necessary rays around to do it with 3 for all of them. So all in all we need $3m + 3n + 4k + 3(n - k) = 3m + 6n + k$ guards. This proves the first direction of the lemma.

Suppose we are given a natural guarding of $\mathcal{P}(G)$ using at most $3m+6n+k$ guards. We say a guard *belongs* to a gadget, if it is a natural vertex guard or a natural edge guard on this gadget. We can find a vertex cover of G as follows. Take all vertices whose punishment gadget has more than 3 guards belonging to it and put them into the cover. Furthermore, add one of the endpoints of all edges whose gadget has more than 3 guards belonging to it. So far we have not put more than k vertices into the cover since there cannot be more than k edge or punishment gadgets with more than 3 guards. (See the observations above. At least 3 guards belong to every vertex gadget and at least 3 guards belong to each of the other $m + n$ gadgets, so there remain at most k guards that can create such edge– or punishment gadgets with 4 or more guards.) Indeed, this set forms a cover of G because every edge that has no endpoint whose gadget is guarded in the first way needs at least 4 guards. Therefore one of its endpoints got added to the cover. $\square$

Outlook. The reduction sketched above shows that it is hard to find an optimal guarding for a collection of polygons or equivalently—looking at the complement—for a polygon with holes. Currently we are working to extend the result to the case where the input consists of a single simple polygon. We also plan to investigate the complexity of finding general, not necessarily natural, guardings.

References

[1] D. Eppstein, M. Goodrich, N. Sitchinava. Guard placement for efficient point-in-polygon proofs. Proc. 23rd SoCG (2007), pp. 27–36.

[2] M. Damian, R. Flatland, J. O'Rourke, S. Ramaswami. A new lower bound on guard placement for wireless localization. 17th FWCG (2007). http://arxiv.org/abs/0709.3554

[3] T. Christ, M. Hoffmann, Y. Okamoto, T. Uno. Improved bounds for the wireless localization problem. Proc. 11th SWAT (2008), pp. 77–89.

[4] R. Uehara. NP-complete problems on a 3-connected cubic planar graph and their applications. Technical Report TWCU-M-0004, Tokyo Woman's Christian University, 1996. http://www.jaist.ac.jp/~uehara/ps/triangle.ps.gz

[5] I. Bárány, G. Rote. Strictly convex drawings of planar graphs. Documenta Mathematica 11 (2006), pp. 369–391.

Shortest Path Problems on a Polyhedral Surface[*]

Atlas F. Cook IV [†] Carola Wenk [†]

Abstract

We develop algorithms to compute shortest path edge sequences, Voronoi diagrams, the Fréchet distance, and the diameter for a polyhedral surface.

1 Introduction

Two questions are invariably encountered when dealing with shortest path problems. The first question is how to represent the combinatorial structure of a shortest path. In the plane with polygonal obstacles, all shortest path vertices occur at obstacle vertices, so a shortest path can be combinatorially described as a sequence of obstacle vertices [8]. On a polyhedral surface, a shortest path need not turn at vertices [11], so a shortest path is often described combinatorially by an *edge sequence* that represents the sequence of edges encountered by the path [1].

The second commonly encountered shortest path question is how to compute shortest paths. Let M be the complexity of a polyhedral surface. On a *convex* polyhedral surface, Mount [12] shows that $\Theta(M^4)$ combinatorially distinct shortest paths exist, and Schevon and O'Rourke [13] show that only $\Theta(M^3)$ of these shortest paths are *maximal* (such paths cannot be extended at either end without creating a suboptimal path). Agarwal et al. [1] use these properties to compute $\Theta(M^4)$ shortest path edge sequences in $O(M^6 2^{\alpha(M)} \log M)$ time, where $\alpha(M)$ is the inverse Ackermann function. They also compute the diameter of a convex polyhedral surface in $O(M^8 \log M)$ time.

We improve the diameter and edge sequence algorithms of Agarwal et al. [1] on a convex polyhedral surface by a linear factor. Our improvement combines star unfolding and kinetic Voronoi diagram techniques. In addition, we achieve a linear factor improvement over the Fréchet distance algorithm of Maheshwari and Yi [10] for polygonal curves on a convex polyhedral surface, and we present the first algorithm to compute the Fréchet distance between polygonal curves on a non-convex polyhedral surface. Our motivation for studying the Fréchet distance on a polyhedral surface is that teaming up two people for safety reasons is common practice in many real-life situations, ranging from scouts in summer camp, to fire fighters and police officers, and even to astronauts exploring the moon. In all of these applications, two team members need to coordinate their movement in order to stay within "walking distance" so that fast assistance can be offered in case of an emergency. The Fréchet distance is an ideal model for this scenario. We summarize our results in Table 1. The following notation is used throughout this paper. Let M be the total complexity of a polyhedral surface and two polygonal curves A and B that lie on the surface. Let $\pi(s,t)$ denote a shortest path on a surface between points s and t, and let $d(s,t)$ be the Euclidean length of $\pi(s,t)$.

2 Shortest Path Problems on a Polyhedral Surface

2.1 Shortest Path Edge Sequences

This section contains superset and exact algorithms for computing the $\Theta(M^4)$ shortest path edge sequences on a convex polyhedral surface $\mathcal{P}$. Both of our algorithms are linear factor improvements over related results of Agarwal et al. [1]. Let $v_1, ..., v_M$ be the corner vertices of $\mathcal{P}$, and let $\Pi = \{\pi(s,v_1), ..., \pi(s,v_M)\}$ be an angularly ordered set of non-crossing shortest paths from a source point $s \in \mathcal{P}$ to each corner vertex $v_j \in \mathcal{P}$. The *star unfolding* $\mathcal{S}$ is a simple polygon [4] defined by cutting $\mathcal{P}$ along each of the shortest paths in Π and unfolding the resulting shape into the plane. Since the source point s touches all M cuts, $s \in \mathcal{P}$ maps to M image points $s_1, ..., s_M$ on the (two dimensional) boundary of the unfolded simple polygon $\mathcal{S}$ (see Figure 1).

An *equator* [5] in the star unfolding is a closed polygonal curve through the points $v_1, ..., v_M, v_1$. The region inside the equator contains no source image and is called the *core* [7]. The disconnected regions outside the core each contain a source image and are collectively referred to as the *anti-core* [7]. A *core edge* is an image of an edge of $\mathcal{P}$ that was never cut during the unfolding process. Each of the $O(M)$ core edges has both of its endpoints at vertices of $\mathcal{S}$ and is entirely contained in the core (see Figure 1b). Each of the $\Theta(M^2)$ *anti-core edges* is an image of an edge of $\mathcal{P}$ that was cut during the unfolding process.

The star unfolding of s can be used to compute $\pi(s,t)$ for points $s,t \in \mathcal{P}$ as follows. The shortest

[*]This work has been supported by the National Science Foundation grant NSF CAREER CCF-0643597.

[†]Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249-0667, USA, acook@cs.utsa.edu, carola@cs.utsa.edu

	Time	Space
Star Unfolding maintained over all edges	$O(M^4)$*	$O(M^4)$
Kinetic Voronoi Diagram maintained over all edges	$O(M^5 2^{\alpha(M)} \log M)$	$O(M^5 2^{\alpha(M)})$
Edge Sequences (superset)	$O(M^5)$	$O(M^5)$
Edge Sequences (exact)	$O(M^5 2^{\alpha(M)} \log M)$	$O(M^4 2^{\alpha(M)})$
Diameter	$O(M^7 \log M)$	$O(M^4)$
Fréchet Distance	$O(M^6 \log^2 M)$ $O(M^7 \log^2 M)$*	$O(M^2)$ $O(M^3)$

Table 1: Summary of our contributions. An asterisk indicates a result for a *non-convex* polyhedral surface.

path $\pi(s,t)$ always originates from one of the source images $s_1, ..., s_M$. If the image of t lies in an anti-core region containing s_i, then the optimal shortest path must originate from s_i (i.e., the shortest paths from t to the remaining source images are suboptimal). If the image of t lies in the core, then the nearest source image is determined with Voronoi diagram techniques.

Agarwal et al. [1] partition the M edges of the convex polyhedral surface $\mathcal{P}$ into $O(M^3)$ line segment *edgelets* such that all source points on an edgelet can be associated with the same combinatorial star unfolding. For each edgelet, a star unfolding is computed, and $O(M^3)$ edge sequences are extracted from it. This yields an $O(M^6)$ superset of the $\Theta(M^4)$ shortest path edge sequences for $\mathcal{P}$ in $O(M^6)$ time and space [1]. Agarwal et al. [1] also show how to compute the exact set of $\Theta(M^4)$ shortest path edge sequences in $O(M^6 2^{\alpha(M)} \log M)$ time. We speed up both of these algorithms by a linear factor.

Theorem 1 *A star unfolding can be maintained as a source point s varies continuously over all M edges of a (possibly non-convex) polyhedral surface in $O(M^4)$ time and space.*

Proof Sketch. The set Π of shortest paths to each corner vertex defines a combinatorial star unfolding. These paths can only change at edgelet endpoints, so Π can be maintained over a discrete set of events. For each change to Π, we update $O(1)$ anti-core regions and possibly all $O(M)$ core edges in the star unfolding. □

Theorem 2 *A superset of the $\Theta(M^4)$ shortest path edge sequences for a convex polyhedral surface $\mathcal{P}$ can be constructed in $O(M^5)$ time and space.*

Proof. Each edgelet defines a star unfolding with source images $s_1, ..., s_M$. For each s_i, construct an edge sequence from s_i to each of the $O(M)$ anti-core edges in the anti-core region for s_i and to each of the $O(M)$ core edges. This yields $O(M^2)$ edge sequences per edgelet, and $O(M^5)$ edge sequences over all edgelets. This is sufficient to determine the desired superset because only core edges have shortest path edge sequences to multiple sites, and this approach considers all possibilities. □

The *exact* set of $\Theta(M^4)$ shortest path edge sequences can be determined with kinetic Voronoi diagram techniques. Albers et al. [2] show that for point sites moving along line segments at constant speeds, each *pair* of sites defines $O(M 2^{\alpha(M)})$ events, and each event is handled in $O(\log M)$ time.

Theorem 3 *A kinetic Voronoi diagram of source images can be maintained in $O(M^4 2^{\alpha(M)} \log M)$ time and $O(M^4 2^{\alpha(M)})$ space as a source point varies continuously over one edge e of a convex polyhedral surface $\mathcal{P}$.*

Proof. A kinetic Voronoi diagram for the *first* edgelet on e has $O(M^2 \cdot M 2^{\alpha(M)})$ events [2] due to the linear motion of $O(M^2)$ *pairs* of source images in the star unfolding. Each of the $O(M^2)$ subsequent edgelets on e can be handled by removing one site (i.e., a source image) and adding a new site. All other sites continue to be parameterized along the same line segments as in the previous edgelet. Thus, each of the $O(M^2)$ edgelets contributes $M-1$ new *pairs* of sites and adds $O(M \cdot M 2^{\alpha(M)})$ new events to the event queue. □

We construct the exact set of shortest path edge sequences for an edgelet α as follows. Maintaining a kinetic Voronoi diagram for α yields a two-dimensional parameterized Voronoi cell φ_i for each source image s_i. The unique edge sequence in the star unfolding's dual graph from s_i to a core edge e represents a shortest path if and only if e intersects φ_i for some $s \in \alpha$.

To decide in logarithmic time whether e intersects φ_i, Agarwal et al.[1] represent each parameterized

Voronoi vertex as an algebraic curve. For a vertex $v_i \in \mathcal{S}$ that touches the anti-core region for the source image s_i, they triangulate the region of the core that is directly visible to v_i such that every triangle Δ has apex v_i. Using polar coordinates centered at v_i, they compute an upper envelope μ of the algebraic curves defining φ_i and partition these curves such that each curve segment is fully contained in one of these triangles. Inside each triangle Δ, the dual graph of the core has at most one degree three vertex because each core edge is a *chord* of Δ. Agarwal et al. [1] compute a dual graph *tree* D_Δ for each triangle in $O(M)$ time.

Lemma 4 *D_Δ can be constructed implicitly in $O(\log M)$ time.*

Proof Sketch. Determine the portion of the core's dual graph D that lies inside Δ by point locating the three vertices of Δ in D. $\square$

For each constant complexity algebraic curve on the upper envelope $\mu \cap \Delta$, Agarwal et al. [1] perform a binary search on the two paths in D_Δ. The deepest edge on each of these two paths that intersects some arc of $\mu \cap \Delta$ defines a *maximal* shortest path edge sequence. The set of all prefixes of these maximal sequences defines $\Theta(M^4)$ shortest path edge sequences.

Theorem 5 *The $\Theta(M^4)$ shortest path edge sequences for a convex polyhedral surface $\mathcal{P}$ with M edges can be constructed in $O(M^5 2^{\alpha(M)} \log M)$ time and $O(M^4 2^{\alpha(M)})$ space.*

Proof. Let n_i be the total number of parameterized Voronoi vertices over all edgelets, and let t_Δ be the time to process each triangle Δ. The technique of Agarwal et al. [1] requires $O(n_i \log M + M^5 t_\Delta)$ time. Since they assume $n_i \in O(M^6 2^{\alpha(M)})$ and $t_\Delta \in O(M)$, they obtain $O(M^6 2^{\alpha(M)} \log M)$ time. By Theorem 3, $n_i \in O(M^5 2^{\alpha(M)})$ over all $O(M)$ edges of $\mathcal{P}$, and by Lemma 4, $t_\Delta \in O(\log M)$ time. $\square$

2.2 Diameter

The *diameter* of a convex polyhedral surface is the largest shortest path distance between any pair of points on the surface. Agarwal et al. [1] compute the diameter in $O(M^8 \log M)$ time by defining $O(M^4)$ *ridge-free regions* on the surface such that all source points in a ridge-free region have the same combinatorial star unfolding.

Theorem 6 *The diameter of a convex polyhedral surface $\mathcal{P}$ with M edges can be constructed in $O(M^7 \log M)$ time and $O(M^4)$ space.*

Proof Sketch. Maintain a kinetic Voronoi diagram over all ridge-free regions. Each of the $O(M^4)$ ridge

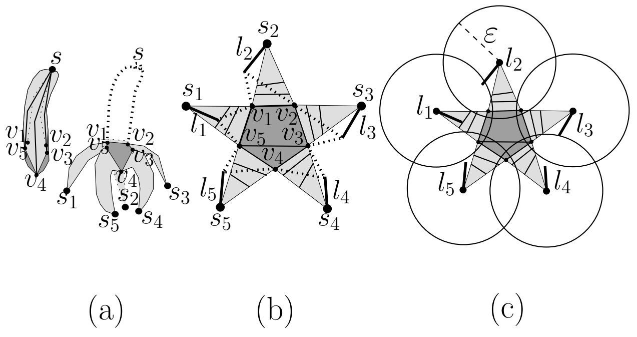


Figure 1: The star unfolding of a polyhedral surface.

free regions defines $O(M^3)$ new events, for a total of $O(M^7)$ parameterized Voronoi vertices. Each vertex can be associated with a function that defines the distance from this vertex to its defining source image. The maximum value defined by these functions is the diameter. $\square$

2.3 Fréchet Distance

The *Fréchet distance* [3] is a similarity metric for *continuous* shapes that is defined for two polygonal curves $A, B : [0,1] \to \mathbb{R}^\mathfrak{d}$ as

$$\delta_F(A,B) = \inf_{\alpha,\beta:[0,1]\to[0,1]} \sup_{t\in[0,1]} d(A(\alpha(t)), B(\beta(t)))$$

where α and β range over continuous non-decreasing reparameterizations, and d is a distance metric for points. For a given constant $\varepsilon \geq 0$, *free space* is defined as $\{(s,t) \mid s \in A,\ t \in B,\ d(s,t) \leq \varepsilon\}$. Let $\delta_C(A,B)$ (resp. $\delta_N(A,B)$) denote the Fréchet distance between polygonal curves A and B on a convex (resp. non-convex) polyhedral surface. Maheshwari and Yi [10] have previously shown how to compute $\delta_C(A,B)$ in $O(M^7 \log M)$ time by enumerating all edge sequences. However, their approach relies on [9] whose key claim "has yet to be convincingly established" [1]. We compute $\delta_C(A,B)$ without enumerating edge sequences in $O(M^6 \log^2 M)$ time and $O(M^2)$ space.

We first describe all free space for a given constant $\varepsilon \geq 0$. The star unfolding $\mathcal{S}$ maps a fixed source point $s \in A$ to a set of source image points $s_1, ..., s_M$ and maps B to a set of core and anti-core edges. *Free space* is defined by the union of a set of disks $d_1, ..., d_M$, where each disk d_i has radius ε and is centered at s_i. As the source point s varies continuously on an edgelet, the core is fixed, and each s_i is parameterized along a line segment l_i in the star unfolding (see Figure 1b).

Theorem 7 *$\delta_C(A,B)$ can be computed in $O(M^6 \log^2 M)$ time and $O(M^2)$ space.*

Proof Sketch. A star unfolding is maintained over all edgelets by Theorem 1. The free space for an *anti-core* edge is defined by the intersection of this edge with one nearest disk d_i. The free space for each *core* edge is defined by the intersection of this edge with $d_1 \cup ... \cup d_M$. After computing all free space, reachability information [3] can be propagated through the free space via plane sweep [6], and parametric search [3] can be applied to a set of algebraic free space vertex curves. □

For a *non-convex* polyhedral surface, the star unfolding can overlap itself [5] but is still defined by a set of shortest paths from the source to every corner vertex [5, 11]. Thus, a core can still be defined by a polygonal equator with $O(M)$ complexity. Since shortest paths only turn at vertices in the star unfolding [11], an anti-core region containing s_i must have an *hourglass* shape [8, 6] because it is bounded by two line segments and two shortest paths in the plane.

Theorem 8 *$\delta_N(A, B)$ can be computed in $O(M^7 \log^2 M)$ time and $O(M^3)$ space.*

Proof Sketch. Maintain a star unfolding over all edgelets. *One* hourglass defines the free space to an anti-core edge, and $O(M)$ hourglasses define the free space to a core edge. After computing all free space, a combination of plane sweep and parametric search techniques yields $\delta_N(A, B)$. □

3 Conclusion

We develop algorithms to compute edge sequences, Voronoi diagrams, the Fréchet distance, and the diameter for a polyhedral surface. Our work speeds up the edge sequence and diameter approaches of Agarwal et al. [1] by a linear factor and introduces many algorithms that apply to convex and non-convex polyhedral surfaces. Future work could attempt to compute the $\Theta(M^4)$ shortest path edge sequences in $\Theta(M^4)$ time. Additional shortest path map and link distance results were omitted from this version due to space concerns.

References

[1] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. A. Schevon. Star unfolding of a polytope with applications. *SIAM Journal on Computing*, 26(6):1689–1713, 1997.

[2] G. Albers, J. S. B. Mitchell, L. J. Guibas, and T. Roos. Voronoi diagrams of moving points. *Journal of Computational Geometry and Applications*, 8:365–380, 1998.

[3] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Journal of Computational Geometry and Applications*, 5:75–91, 1995.

[4] B. Aronov and J. O'Rourke. Nonoverlap of the star unfolding. *7th Symposium on Computational Geometry (SoCG)*, pages 105–114, 1991.

[5] J. Chen and Y. Han. Shortest paths on a polyhedron. *Journal of Computational Geometry and Applications*, 6(2):127–144, 1996.

[6] A. F. Cook IV and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *25th Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008.

[7] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, New York, NY, USA, 2007.

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

[9] Y.-H. Hwang, R.-C. Chang, and H.-Y. Tu. Finding all shortest path edge sequences on a convex polyhedron. *Workshop on Algorithms and Data Structures (WADS)*, 1989.

[10] A. Maheshwari and J. Yi. On computing Fréchet distance of two paths on a convex polyhedron. *21st European Workshop on Computational Geometry (EuroCG)*, 2005.

[11] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[12] D. M. Mount. The number of shortest paths on the surface of a polyhedron. *SIAM Journal on Computing*, 19(4):593–611, 1990.

[13] C. Schevon and J. O'Rourke. The number of maximal edge sequences on a convex polytope. *26th Allerton Conference on Communication, Control, and Computing*, pages 49–57, 1988.

Conflict-Free Coloring Made Stronger

Elad Horev* Roi Krakovski† Shakhar Smorodinsky‡

Abstract

In this paper we prove the following:

(i) Any set of n discs in the plane can be colored with a total of at most $O(k \log n)$ colors such that for any point p that is covered by at least k discs, there are at least k distinct discs each of which is colored by a color distinct from all other discs containing p and for any point p covered by at most k discs, all discs covering p are colored distinctively. We call such a coloring a *k-Strong Conflict-Free* coloring. We extend this result to pseudo-discs and arbitrary regions with linear union-complexity.

(ii) More generally, for families of n simple closed Jordan regions with union-complexity bounded by $O(n^{1+\alpha})$, we prove that there exists a k-strong conflict-free coloring with at most $O(kn^{\alpha})$ colors.

(iii) We prove that any set of n axis-parallel rectangles can be k-strong conflict-free colored with at most $O(k \log^2 n)$ colors.

(iv) For the case of discs and axis-parallel rectangles, we provide a lower bound of $\Omega(\frac{k}{\log k} \log n)$ on the maximum number of colors needed in k-strong conflict-free coloring in the worst case.

(v) We provide a general framework for k-strong conflict-free coloring arbitrary hypergraphs. This framework relates the notion of k-strong conflict-free coloring and the recently studied notion of k-colorful coloring.

All of our proofs are constructive. That is, there exist polynomial time algorithms for computing such colorings.

1 Introduction and Preliminaries

Motivated by modeling frequency assignment to cellular antennae, Even et al. [14] introduced the notion of Conflict-Free colorings. A *Conflict-Free* coloring (CF in short) of a hypergraph $H = (V, \mathcal{E})$ is a coloring of the vertices V such that for any non-empty hyperedge $e \in \mathcal{E}$ there is some vertex $v \in e$ whose color is distinct from all other colors of vertices in e. For a hypergraph H, one seeks the least number of colors l such that there exists an l-coloring of H which is conflict-free. Let $\mathcal{R}$ be a finite collection of regions in $\mathbb{R}^d$, $d \geq 1$. For a point $p \in \mathbb{R}^d$, define $r(p) = \{R \in \mathcal{R} : p \in R\}$. The hypergraph $(\mathcal{R}, \{r(p)\}_{p \in \mathbb{R}^d})$, denoted $H(\mathcal{R})$, is called the hypergraph *induced* by $\mathcal{R}$. Such hypergraphs are referred to as *geometrically induced* hypergraphs. The paper [14] focused on hypergraphs that are induced by geometric objects such as discs, squares etc. Their motivation was a modeling of frequency assignment to cellular antennae in a manner that reduces the spectrum of frequencies used by a network of antennae. The notion of CF-coloring has caught much scientific attention in recent years both from the algorithmic and combinatorial point of view [2, 3, 5, 6, 7, 9, 10, 15, 18].

Our Contribution: We study the notion of *k-strong-conflict-free* (abbreviated, *$kSCF$*) colorings of hypergraphs. This notion extends the notion of *CF*-colorings of hypergraphs.

Definition 1 (k-strong conflict-free coloring:) *Let $H = (V, \mathcal{E})$ be a hypergraph and let $k \in \mathbb{N}$ be some fixed integer. A coloring of V is called* k-strong-conflict-free *for H if for every hyperedge $e \in \mathcal{E}$ with $|e| \geq k$ there exists at least k vertices in e, whose colors are unique among the colors assigned to the vertices of e and for each hyperedge $e \in \mathcal{E}$ with $|e| < k$ all vertices in e get distinct colors . Let $f_H(k)$ denote the least integer l such that H admits a $kSCF$-coloring with l colors.*

Abellanas et al. [1] were the first to study kSCF-coloring[1]. They focused on the special case where V is a finite set of points in the plane and $\mathcal{E}$ consist of all subsets of V which can be realized as an intersection of V with a disc. They showed that in this case the hypergraph admits a kSCF-coloring with $O(\frac{\log n}{\log \frac{ck}{ck-1}})$ $(= O(k \log n))$ colors, for some absolute constant c.

k-colorful coloring The following notion was recently introduced and studied by Aloupis et al. [4] for the special case of hypergraphs induced by discs:

Definition 2 *Let $H = (V, \mathcal{E})$ be a hypergraph, and let φ be a coloring of H. A hyperedge $e \in \mathcal{E}$ is said to be* k-colorful *with respect to φ if there exist k vertices in e that are colored distinctively under φ. The*

*Computer Science department, Ben-Gurion University, Beer Sheva, Israel; horevel@cs.bgu.ac.il.

†Computer Science department, Ben-Gurion University, Beer Sheva, Israel; roikr@cs.bgu.ac.il.

‡Mathematics department, Ben-Gurion University, Beer Sheva, Israel; http://www.math.bgu.ac.il/∼shakhar/ ; shakhar@math.bgu.ac.il.

[1]They referred to such a coloring as k-conflict-free coloring.

coloring φ is called k-colorful *if every hyperedge $e \in \mathcal{E}$ is $\min\{|e|, k\}$-colorful. Let $c_H(k)$ denote the least integer l such that H admits a k-colorful coloring with l colors.*

Aloupis et al. were motivated by a problem related to battery lifetime in sensor networks. See [4, 8, 17] for additional details on the motivation and related problems.

Remark: Every $kSCF$-coloring of a hypergraph H is a k-colorful coloring of H. However, the opposite claim is not necessarily true.

We study a connection between k-colorful coloring and strong-conflict-free coloring of hypergraphs. We show that if a hypergraph H admits a k-colorful coloring with a "small" number of colors (hereditarily) then it also admits a $(k-1)$SCF-coloring with a "small" number of colors. The interrelation between the quoted terms is provided in theorems 1 and 2 below:

Let $H = (V, \mathcal{E})$ be a hypergraph and let $V' \subset V$. We write $H[V']$ to denote the sub-hypergraph of H induced by V', i.e., $H[V'] = (V', \mathcal{E}')$ and $\mathcal{E}' = \{e \cap V' | e \in \mathcal{E}\}$. We write $n(H)$ to denote the number of vertices of H.

Theorem 1 *Let $H = (V, \mathcal{E})$ be a hypergraph with n vertices, and let $k, \ell \in \mathbb{N}$ be fixed integers, $k \geq 2$. If every induced sub-hypergraph $H' \subseteq H$ satisfies $c_{H'}(k) \leq \ell$, then $f_H(k-1) \leq \log_{1+\frac{1}{\ell-1}} n = O(l \log n)$.*

Theorem 2 *Let $H = (V, \mathcal{E})$ be a hypergraph with n vertices, and let $k \geq 2$ be a fixed integer. let $0 < \alpha \leq 1$ be a fixed real. If every induced sub-hypergraph $H' \subseteq H$ satisfies $c_{H'}(k) = O(kn(H')^\alpha)$, then $f_H(k-1) = O(kn^\alpha)$.*

We provide an upper bound on the number of colors required by $kSCF$-coloring of geometrically induced hypergraphs as a function of the union-complexity of the regions that induce the hypergraphs. Below we describe the relations between the union-complexity of the regions, k-colorful and $(k-1)$SCF coloring of the underlying hypergraph. First, we need to define the notion of union-complexity.

Definition 3 *For a family $\mathcal{R}$ of n simple closed Jordan regions in the plane, let $\partial\mathcal{R}$ denote the boundary of the union of the regions in $\mathcal{R}$. The* union-complexity *of $\mathcal{R}$ is the number of intersection points, of a pair of boundaries of regions in $\mathcal{R}$, that belong to $\partial\mathcal{R}$.*

For a set $\mathcal{R}$ of n simple closed planar Jordan regions, let $\mathcal{U}_\mathcal{R} : \mathbb{N} \to \mathbb{N}$ be a function such that $\mathcal{U}_\mathcal{R}(m)$ is the maximum union-complexity of any subset of k regions in $\mathcal{R}$ over all $k \leq m$, for $1 \leq m \leq n$. We abuse the definition slightly and assume that the union-complexity of any set of n regions is at least n. When dealing with geometrically induced hypergraphs, we consider k-colorful coloring and $kSCF$-coloring of hypergraphs that are induced by simple closed Jordan regions having union-complexity at most $O(n^{1+\alpha})$, for some fixed parameter $0 \leq \alpha \leq 1$. The value $\alpha = 0$ corresponds to regions with linear union-complexity such as discs or pseudo-discs (see, e.g., [16]). The value $\alpha = 1$ corresponds to regions with quadratic union-complexity. See [12, 13] for additional families with sub-quadratic union-complexity.

Theorem 3 *Let $k \geq 2$, let $0 \leq \alpha \leq 1$, and let c be a fixed constant. Let $\mathcal{R}$ be a set of n simple closed Jordan regions such that $\mathcal{U}_\mathcal{R}(m) \leq cm^{1+\alpha}$, for $1 \leq m \leq n$, and let $H = H(\mathcal{R})$. Then $c_H(k) = O(kn^\alpha)$.*

Combining Theorem 1 with Theorem 3 (for $\alpha = 0$) and Theorem 2 with Theorem 3 (for $0 < \alpha < 1$) yields the following result:

Theorem 4 *Let $k \geq 2$, let $0 \leq \alpha \leq 1$, and let c be a constant. Let $\mathcal{R}$ be a set of n simple closed Jordan regions such that $\mathcal{U}_\mathcal{R}(m) = cm^{1+\alpha}$, for $1 \leq m \leq n$. Let $H = H(\mathcal{R})$. Then:*

$$f_H(k-1) = \begin{cases} O(k \log n), & \alpha = 0, \\ O(kn^\alpha), & 0 < \alpha \leq 1. \end{cases}$$

We consider $kSCF$-colorings of hypergraphs induced by axis-parallel rectangles in the plane. Obviously, axis-parallel rectangles might have quadratic union-complexity, for example, by considering a grid-like construction of $n/2$ disjoint (horizontally narrow) rectangles and $n/2$ disjoint (vertically narrow) rectangles. For a hypergraph H induced by axis-parallel rectangles, Theorem 4 states that $f_H(k-1) = O(kn)$. This bound is meaningless, since the bound $f_H(k-1) \leq n$ is trivial. Nevertheless, we provide a near-optimal upper bound for this case in the following theorem:

Theorem 5 *Let $k \geq 2$. Let $\mathcal{R}$ be a set of n axis-parallel rectangles, and let $H = H(\mathcal{R})$. Then $f_H(k-1) = O(k \log^2 n)$.*

In order to obtain Theorem 5 we prove the following theorem:

Theorem 6 *Let $H = H(\mathcal{R})$, be the hypergraph induced by a family $\mathcal{R}$ of n axis-parallel rectangles in the plane, and let $k \in \mathbb{N}$ be an integer, $k \geq 2$. For every induced sub-hypergraph $H' \subseteq H$ we have: $c_{H'}(k) \leq k \log n$.*

The proof of Theorem 5 is therefore an easy consequence of Theorem 6 combined with Theorem 1

Har-Peled and Smorodinsky [15] proved that any family $\mathcal{R}$ of n axis-parallel rectangles admit a CF-coloring with $O(\log^2 n)$ colors. Their proof uses the probabilistic method. They also provide a randomized algorithm for obtaining CF-coloring with at most $O(\log^2 n)$ colors. Later, Smorodinsky [18] provided a deterministic polynomial-time algorithm that produces a CF-coloring for n axis-parallel rectangles with $O(\log^2 n)$ colors. Theorem 5 thus generalizes the results of [15] and [18].

Finally, it is shown that the upper bounds provided in Theorem 4 for $\alpha = 0$ and Theorem 5 are near optimal. Specifically, we provide lower bounds on the number of colors required by any $kSCF$-coloring of hypergraphs induced by (unit) discs and axis-parallel squares in the plane:

Theorem 7 *(i) There exist families $\mathcal{R}$ of n (unit) discs for which $f_{H(\mathcal{R})}(k) = \Omega(\frac{k}{\log k}\log n)$*

(ii) There exist families $\mathcal{R}$ of n axis-parallel squares for which $f_{H(\mathcal{R})}(k) = \Omega(\frac{k}{\log k}\log n)$.

All of our proofs are constructive. In other words, there exist deterministic polynomial-time algorithms to obtain the required $kSCF$ coloring with the promised bounds. In this abstract, we omit the technical details of all the underlying proofs and algorithms due to space limitations.

2 A Framework For Strong-Conflict-Free Coloring

In this section we prove that if there exist fixed integers k and l such that an n-vertex hypergraph H admits the hereditary property that every vertex-induced sub-hypergraph H' of H admits a k-colorful coloring with at most l colors, then H admits a $(k-1)SCF$-coloring with $O(l \log n)$ colors. For the case when l is replaced with the function $kn(H')^{\alpha}$ we get a better bound without the $\log n$ factor. Theorems 1 and 2 establish the mentioned framework. The proof is constructive. The following framework produces a valid $(k-1)$SCF coloring for a hypergraph H satisfying the conditions of Theorems 1 and 2.
Framework $\mathcal{A}$:
Input: A hypergraph H satisfying the conditions of Theorems 1 and 2.
Output: A $(k-1)SCF$-coloring of H.

```
1: $i \leftarrow 1$ {$i$ denotes an unused color.}
2: **while** $V \neq \emptyset$ **do**
3:     **Auxiliary Coloring:** Let $\varphi : V \to [\ell]$ be a $k$-colorful coloring of $H[V]$ with at most $\ell$ colors.
4:     Let $V'$ be a color class of $\varphi$ of maximum cardinality.
5:     **Color:** Set $\chi(u) = i$ for every vertex $u \in V'$.
6:     **Discard:** $V \leftarrow V \setminus V'$.
7:     **Increment:** $i \leftarrow i + 1$.
8: **end while**
9: **Return** $\chi$.
```

Let χ denote the coloring obtained by appllying the framework $\mathcal{A}$ on H. The number of colors used by χ is the number of iterations performed by $\mathcal{A}$. By the pigeon-hole principle, at least $|V|/\ell$ vertices are removed in each iteration (where V is the set of vertices remained after the last iteration). Therefore, the total number of iterations performed by $\mathcal{A}$ is bounded by $\log_{1+\frac{1}{\ell-1}} n$. Thus, the coloring χ uses at most $\log_{1+\frac{1}{\ell-1}} n$ colors. If in step 3 of the framework, l is replaced with the function $k|V|^{\alpha}$ (for a fixed parameter $0 < \alpha < 1$), then by the pigeon-hole principle at least $\frac{|V|^{1-\alpha}}{k}$ vertices of H are discarded in step 6 of that iteration. It is easily seen that the number of iterations performed in this case is bounded by $O(kn^{\alpha})$ where $n = n(H)$.

Next, we prove that the coloring χ is indeed a $(k-1)SCF$-coloring of H. The colors of χ are the indices of iterations of $\mathcal{A}$. Let $e \in \mathcal{E}$ be a hyperedge of H. If $|e| \leq k$ then it is easily seen that all colors of vertices of e are distinct. Indeed, by the property of the auxiliary coloring φ in step 3 of the framework, every vertex of e is colored distinctively and in each such iteration, at most one vertex from e is colored by χ so χ colors all vertices of e in distinct iterations. Next, assume that $|e| > k$. We prove that e contains at least $k-1$ vertices that are assigned unique colors in χ. For an integer r, let $\{\alpha_1, \ldots, \alpha_r\}$ denote the r largest colors in decreasing order that are assigned to some vertices of e. That is, the color α_1 is the largest color assigned to a vertex of e, the color α_2 is the second largest color and so on. In what follows, we prove a stronger assertion that for every $1 \leq j \leq k-1$ the color α_j exists and is unique in e. The proof is by induction on j. α_1 exists in e by definition. For the base of the induction we prove that α_1 is unique in e. Suppose that the color α_1 is assigned to at least two vertices $u, v \in e$, and consider iteration α_1 of $\mathcal{A}$. Let $H' = H[\{x \in V : \chi(x) \geq \alpha_1\}]$, and let φ be the k-colorful coloring obtained for H' in step 3 of iteration α_1. Put $e' = \{x \in e : \chi(x) \geq \alpha_1\}$. $e' \subset e$ is a hyperedge in H'. Since $u, v \in e'$ then $|e| \geq 2$. φ is k-colorful for H' so e' contains at least two vertices that are colored distinctively in φ. In iteration α_1, the vertices of one color class of φ are removed from e'. Since e' contains vertices from two color classes of φ, it follows that after iteration α_1 at least one vertex of e' remains. Thus, at least one vertex of e' is colored in a later iteration than α_1, a contradiction to the maximality of α_1. The induction hypothesis is that in χ the colors $\alpha_1, \ldots, \alpha_{j-1}$, $1 < j \leq k-1$, all exist and are unique in the hyperedge e. Consider the color α_j. There exists a vertex $u \in e$ such that $\chi(u) = \alpha_j$; for otherwise it follows from the induction hypothesis that $|e| < k-1$ since the colors $\alpha_1, \ldots, \alpha_{j-1}$ are all unique in e and

$j-1 < k-1$. We prove that the color α_j is unique in e. Assume to the contrary that α_j is not unique at e, and that in χ the color α_j is assigned to at least two vertices $u, v \in e$. Put $H'' = H[\{u \in V : \chi(u) \geq \alpha_j\}]$, and let φ'' be the k-colorful coloring obtained for H'' in step 3 of iteration α_j. Put $e'' = \{u \in e : \chi(u) \geq \alpha_j\}$. e'' is a hyperedge of H''. By the induction hypothesis and the definition of the colors $\alpha_1, \ldots, \alpha_{j-1}$, after iteration α_j a set $U \subset e''$ of exactly $j-1$ vertices of e'' remains. In addition, $u, v \in e''$ and $U \cap \{u, v\} = \emptyset$. Consequently, $|e''| \geq j+1$. Since φ'' is k-colorful then e'' contains vertices from $\min\{k, j+1\}$ color classes of φ''. $j \leq k-1$ so $\min\{k, j+1\} = j+1$. Since in iteration α_j the vertices of one color class of φ'' are removed from e'', it follows that after iteration α_j at least j vertices of e'' remain. This is a contradiction to the induction hypothesis. This completes the proof of the theorem.

Theorems 1 and 2 asserts that in order to attain upper bounds on $f_H(k)$, for a hypergraph H, one may concentrate on attaining a bound on $c_H(k)$. Given a k-colorful coloring of H, the framework $\mathcal{A}$ obtains a strong-conflict-free coloring of H in a constructive manner. In this paper, computational efficiency is not of main interest. However, it can be seen that for certain families of geometrically induced hypergraphs, framework $\mathcal{A}$ produces an efficient algorithm. In particular, for hypergraphs induced by discs or axis-parallel rectangles, framework $\mathcal{A}$ produces an algorithm with a low degree polynomial running time. Colorful-colorings of such hypergraphs can be computed once the arrangement of the discs is computed together with the depth of every face.

References

[1] M. Abellanas, P. Bose, J. Garcia, F. Hurtado, M. Nicolas, and P. A. Ramos. On properties of higher order delaunay graphs with applications. In *EWCG*, 2005.

[2] D. Ajwani, K. Elbassioni, S. Govindarajan, and S. Ray. Confict-free coloring for rectangle ranges using $\tilde{O}(n^{.382+\epsilon})$ colors. In *SPAA '07: Proc. 19th ACM Symp. on Parallelism in Algorithms and Architectures*, pages 181–187, 2007.

[3] N. Alon and S. Smorodinsky. Conflict-free colorings of shallow discs. In *SoCG '06: Proc. 22nd Annual ACM Symposium on Computational Geometry*, pages 41–43, 2006.

[4] G. Aloupis, J. Cardinal, S. Collette, S. Langerman, and S. Smorodinsky. Coloring geometric range spaces. In *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, pages 146–157, Búzios, Brazil, 2008.

[5] A. Bar-Noy, P. Cheilaris, S. Olonetsky, and S. Smorodinsky. Online conflict-free colorings for hypergraphs. In *ICALP*, pages 219–230, 2007.

[6] A. Bar-Noy, P. Cheilaris, S. Olonetsky, and S. Smorodinsky. Weakening the online adversary just enough to get optimal conflict-free colorings for intervals. In *SPAA*, pages 194–195, 2007.

[7] A. Bar-Noy, P. Cheilaris, and S. Smorodinsky. Conflict-free coloring for intervals: from offline to online. In *SPAA '06: Proceedings of The Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 128–137, New York, NY, USA, 2006. ACM Press.

[8] A. L. Buchsbaum, A. Efrat, S. Jain, S. Venkatasubramanian, and K. Yi. Restricted strip covering and the sensor cover problem. In *SODA*, pages 1056–1063, 2007.

[9] K. Chen, A. Fiat, M. Levy, J. Matoušek, E. Mossel, J. Pach, M. Sharir, S. Smorodinsky, U. Wagner, and E. Welzl. Online conflict-free coloring for intervals. *SIAM J. Comput.*, 36:545–554, 2006, (See also in Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms, 2005).

[10] K. Chen, H. Kaplan, and M. Sharir. Online conflict-free coloring for halfplanes, congruent disks, and axis-parallel rectangles, (manuscript), 2005.

[11] X. Chen, J. Pach, M. Szegedy, and G. Tardos. Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. In *SODA*, pages 94–101, 2008.

[12] A. Efrat. The complexity of the union of (α, β)-covered objects. In *"Proc. 15th Annu. ACM Sympos. Comput. Geom."*, pages 134–142, 1999.

[13] A. Efrat and M. Sharir. On the complexity of the union of fat convex objects in the plane. *Discrete and Comput.Geom.*, 23:171–189, 2000.

[14] G. Even, Z. Lotker, D. Ron, and S. Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. *SIAM J. Comput.*, 33:94–136, 2003, (See also in Proc. 43rd Annual Symposium on Foundations of Computer Science, 2002).

[15] S. Har-Peled and S. Smorodinsky. On conflict-free coloring of points and simple regions in the plane. *Discrete and Comput.Geom.*, pages 47–70, 2005, (See also in 19th Annual Symposium on Computational Geometry, 2003).

[16] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[17] J. Pach and G. Tóth. Decomposition of multiple coverings into many parts. In *Symposium on Computational Geometry*, pages 133–137, 2007.

[18] S. Smorodinsky. On the chromatic number of some geometric hypergraphs. *SIAM Journal on Discrete Mathematics*, 21:676–687, 2007, (See also in Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms, 2006).

Two Applications of Point Matching

Günter Rote*

Abstract

The two following problems can be solved by a reduction to a minimum-weight bipartite matching problem (or a related network flow problem):

a) Floodlight illumination: We are given n infinite wedges (sectors, spotlights) that can cover the whole plane when placed at the origin. They are to be assigned to n given locations (in arbitrary order, but without rotation) such that they still cover the whole plane. (This extends results of Bose et al. [4] from 1997.)

b) Convex partition: Partition a convex m-gon into m convex parts, each part containing one of the edges and a given number of points from a given point set. (García and Tejel 1995 [5], Aurenhammer 2008 [3])

1 The Semi-Assignment Problem

We are given an $m \times n$ *cost matrix* (c_{ij}) and an integer *demand vector* $(n_1, \ldots, n_m)$ with $\sum n_j = n$. The semi-assignment problem is the following optimization problem:

$$\text{minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum\nolimits_{j=1}^{n} x_{ij} = 1, \text{ for } i = 1, \ldots, n \tag{2}$$

$$\sum\nolimits_{i=1}^{m} x_{ij} = n_j, \text{ for } j = 1, \ldots, m \tag{3}$$

$$0 \le x_{ij} \le 1 \tag{4}$$

By network flow theory, there is an optimal solution with $x_{ij} \in \{0,1\}$, and it represents an assignment where each row is assigned to exactly one column, and each column j has n_j rows assigned to it. The special case where $m = n$ and all $n_j = 1$ is the usual *assignment problem.*

Kennington and Wang [6] showed that it can be solved in $O(mn^2)$ time by the shortest augmenting path method.

By linear programming duality (in particular, network flow duality), we get the following characterization of optimal semi-assignments.

Lemma 1 *A feasible 0-1-solution* (x_{ij}) *is an optimal solution of the problem* (1–4) *if and only if any of the following two* (*equivalent*) *conditions holds.*

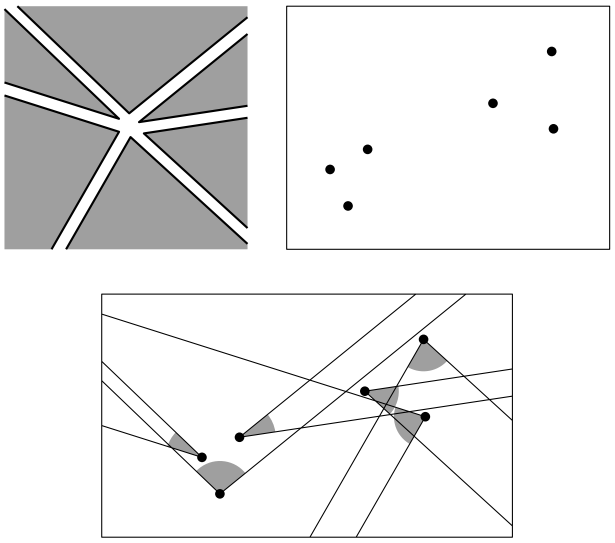


Figure 1: The illumination problem

(i) *There are dual variables u_i and v_j for which complementary slackness holds:*

$$c_{ij} \ge u_i + v_j, \text{ for all } i, j, \tag{5}$$

and $x_{ij} = 1$ implies $c_{ij} = u_i + v_j$.

(ii) *There are variables v_j, such that each row i is assigned to some row j for which $c_{ij} - v_j$ is smallest.*

Proof. Part (i) is the complementary slackness condition of linear programming duality. For part (ii), observe that for given v_j, we must have $u_i \le \min_j(c_{ij} - v_j)$ in order to fulfill (5). It is no loss of generality to set u_i to this maximal possible value $\min_j(c_{ij} - v_j)$, and then $c_{ij} = u_i + v_j$ is fulfilled precisely for those column indices j for which the minimum is achieved. □

We will now give two geometric applications of this lemma. In each case, we have a lower envelope of planes in three dimensions. We want to adjust the planes such that each cell in the projection of the lower envelope contains a prescribed number of points from a given set S.

2 Illumination

We are given n *floodlights* $W_1, \ldots, W_n$, each covering an (infinite) wedge of the plane, and n locations $\mathbf{p}_1, \ldots, \mathbf{p}_n$ for the floodlights, as shown in the upper

*Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany, `rote@inf.fu-berlin.de`

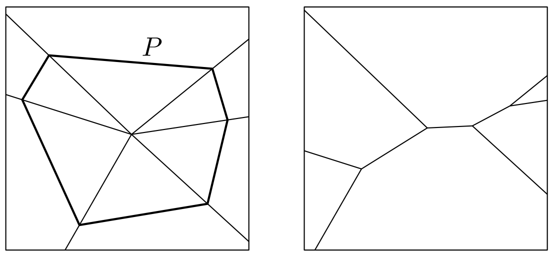


Figure 2: Weighted Voronoi diagrams

part of Figure 1. The task is to find a one-to-one assignment from the floodlights to the locations so that the floodlights collectively illuminate the whole plane, as shown in the lower part.

Bose et al. [4] have shown that this is always possible if the floodlights can be freely rotated, provided that the total angle is at least 2π. We strengthen this result to the case when rotation is not allowed.

Theorem 2 *Any set of n wedges, each of opening angle $< \pi$, which collectively cover the plane when placed at the origin, can be assigned to n given locations such that they still cover the plane.*

The condition that the floodlights must cover the whole range of directions is clearly necessary.

Proof. We place the wedges at the origin and form a convex polygon P with one vertex on every ray, see Figure 2a. Now we lift P to the plane $z = 1$ in three dimensions and form a convex cone through P with vertex at the origin. Figure 2a can be seen as the vertical projection of this cone. Each wedge W_j is represented by a plane π_j with equation $z = \mathbf{a}_j \cdot \mathbf{x}$, where $\mathbf{a}_j$ and $\mathbf{x} = \binom{x}{y}$ are vectors in the plane, and "$\cdot$" denotes the scalar product. The cone is the upper envelope of the planes.

If we shift the planes vertically by adding a constant v_j to each equation:

$$\pi_j : \quad z = \mathbf{a}_j \cdot \mathbf{x} + v_j,$$

the upper envelope becomes a convex polyhedral surface, whose projection on the x-y-plane is a *weighted Voronoi diagram* (or *power diagram*) of the weighted points $\mathbf{a}_j$ with weights v_j, see Figure 2b for an example. We denote the regions in this diagram by R_j.

Lemma 3 *If the weighted Voronoi diagram for some appropriate set of weights v_j contains one point $\mathbf{p}_i$ in each Voronoi cell, then, placing each wedge at the corresponding point will make all wedges disjoint. The opposite wedges (rotated by $180°$) will cover the whole plane.*

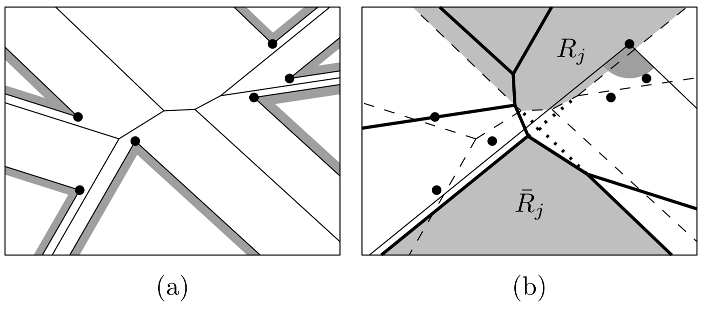


Figure 3: (a) Disjoint wedges in the weighted Voronoi diagram. (b) The furthest-point weighted Voronoi diagram (highlighted) is overlaid over the weighted Voronoi diagram, showing why the whole plane is covered.

Proof. The first statement is obvious, since each wedge W_j is contained in its cell R_j, see Figure 3a.

To prove the second statement, we look at the *lower* envelope of the planes (the *furthest-point* weighted Voronoi diagram), see Figure 3b. One can see that placing the rotated wedge on any point in the region R_j is sufficient to cover the corresponding region $\bar{R}_j$ in the furthest-point weighted Voronoi diagram. In fact, R_j and $\bar{R}_j$ share the lines through their unbounded edges, but they lie on opposite sides of them. Since each cell $\bar{R}_j$ is covered, it follows that the whole plane is covered by all rotated wedges. $\square$

We remark that the condition that the opposite wedges are disjoint is neither necessary for a covering, as witnessed by the example in Figure 1, nor is it in itself sufficient to guarantee a covering.

The lemma will thus solve our problem, except that all wedges are rotated by $180°$. So we only have to start the procedure with the opposite (rotated) set of wedges, in order to get the desired result in the end.

To conclude the proof of the theorem, we have to show that there exists a weighted Voronoi diagram with one point $\mathbf{p}_i$ in each region, as required by Lemma 3.

This follows from a result of Aurenhammer et al. [2], of which we cite a special case:

Theorem 4 *Given n points $\mathbf{p}_i$ and n "slopes" $\mathbf{a}_j$, there is a set of weights v_j such that the weighted Voronoi diagram contains one point per region.*

The connection between Theorem 4 and matchings was noted by Aurenhammer et al. [2]. We use it to give a self-contained proof of Theorem 4 from Lemma 1.

Proof. We solve the assignment problem with $c_{ij} := -\mathbf{a}_i \cdot \mathbf{p}_j$. According to Lemma 1, we get weights v_j such that each point $\mathbf{p}_i$ is assigned to the plane j for which $c_{ij} - v_j = -\mathbf{a}_j \cdot \mathbf{p}_i - v_j$ is smallest, or equivalently, for which $\mathbf{a}_j \cdot \mathbf{p}_i + v_j$ is largest. This is indeed the plane forming the upper envelope at the point $\mathbf{a}_i$, and thus $\mathbf{a}_i$ lies in region j. $\square$

The running time of the algorithm implied by the previous result is dominated by the running time of computing the assignment. The classical network flow methods take $O(n^3)$ time. However, as observed in [2], the objective function $c_{ij} = -\mathbf{a}_i \cdot \mathbf{p}_j$ is equivalent to the objective function $c_{ij} = \|\mathbf{a}_i - \mathbf{p}_j\|^2$, apart from an additive constant. Thus, our problem is equivalent to least-squares bipartite point matching. For point matching with Euclidean lengths as distances (without squaring), Agarwal et al. [1] gave an algorithm of running time $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. It is likely that this algorithm can be extended to our situation.

Theorem 2 can be generalized to higher dimensions, provided that the floodlights come from a polytope in the same way as the polygon P of Figure 2a is associated to the wedges in the plane.

One can give a three-dimensional example showing that the generalization is not true without any additional condition.

It is an open question to decide whether a given three-dimensional polyhedral fan (a set of floodlights) can be assigned to a given set of locations, forming a covering of the plane. It is another open question to characterize the three-dimensional polyhedral fans which can be assigned to an arbitrary set of location.

In higher dimensions, there is no known geometric algorithm that would beat the $O(n^3)$ bound.

3 Convex Partitioning

We are given n points S in a convex m-gon P with sides $e_1, \ldots, e_m$, and m integers b_i with $b_1 + \cdots + b_m = n$. The task is to partition the points into subsets $S = S_1 \cup \cdots \cup S_m$ such that $|S_j| = m_j$ and the convex hulls of $S_j \cup e_j$ are disjoint, see Figure 4a. García and Tejel [5] showed that such a partition always exists, and they gave an $O(n \log n)$ divide-and-conquer algorithm for constructing it.

A particular type of convex partition was recently introduced by Aurenhammer [3]: Consider the polygon P in the x-y-plane of three-dimensional space, and pass a plane π_j through each side e_j, rising above the polygon. The lower envelope of these planes is projected on the x-y-plane and induces a convex partition, which is called a *weighted skeleton*, see Figure 4b. Aurenhammer [3] has shown that there exists always such a partition for which the parts contain the required number of points, and the resulting partition of S is unique (apart from degenerate cases).

We give a different proof by a straightforward reduction to the semi-assignment problem: Let d_{ij} be the distance between point i and the line through

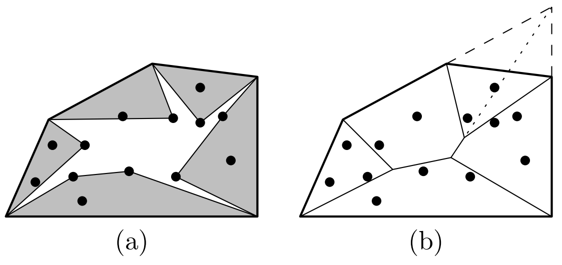


Figure 4: Partitioning points in a convex polygon. (a) an arbitrary convex partition; (b) a weighted skeleton. The dotted lines indicate an instance of the geometric restrictions which hold for weighted skeleta.

e_j. We then solve the semi-assignment problem with $c_{ij} := \log d_{ij}$. By Lemma 1 there are numbers v_j such that each point i is assigned to the side j for which $c_{ij} - v_j$ is smallest. If we take $a_j := \exp v_j$ as the slope of the plane π_j, we get that each point i is assigned to the plane for which d_{ij}/a_j is smallest; this is indeed the plane which forms the lower envelope at point i. $\square$

Aurenhammer sketched an algorithm which takes $O(mn \log n(m + \log n))$ time and $O(m + n)$ space, whereas the semi-assignment algorithm of [6], without taking into account the geometric structure of the problem and the special structure of the cost matrix (c_{ij}), takes $O(mn^2)$ time and $O(m + n)$ space. It remains to be seen whether this matching problem fits into the framework of the matching problems considered in [1], which might lead to an algorithm of running time $O(n^{7/3+\varepsilon})$, for any $\varepsilon > 0$.

References

[1] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir, *Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications.* SIAM J. Comput. 29(3): 912–953 (1999).

[2] Franz Aurenhammer, Friedrich Hoffmann, and Boris Aronov, *Minkowski-type theorems and least-squares clustering.* Algorithmica 20: 61–76 (1998)

[3] Franz Aurenhammer. *Weighted skeletons and fixed-share decomposition.* Int. J. Comput. Geometry Appl. 40: 93–101 (2008)

[4] Prosenjit Bose, Leonidas J. Guibas, Anna Lubiw, Mark H. Overmars, Diane L. Souvaine, Jorge Urrutia. *The floodlight problem.* Int. J. Comput. Geometry Appl. 7(1/2): 153–163 (1997)

[5] A. García and J. Tejel. *Dividiendo una nube de puntos en regiones convexas.* In: Actas VI Encuentros de Geometría Computational, Barcelona, Spain, 1995, pp. 169–174.

[6] J. Kennington and Z. Wang. *A shortest augmenting path algorithm for the semi-assignment problem.* Operations Research 40(1), 178–187 (1992).

Distance k-Sectors and Zone Diagrams

Keiko Imai[*] Akitoshi Kawamura[†] Jiří Matoušek[‡] Yu Muramatsu[§] Takeshi Tokuyama[¶]

Abstract

We prove the existence and uniqueness of the zone diagram of a given set of sites in Euclidean space. This was known for point sites in the plane, but our proof is simpler even for this specific case. We also show the existence of a distance k-sector between two sites. Both proofs rely on the Knaster–Tarski theorem on fixed points of monotone functions.

1 Introduction

Geometric bisection is a fundamental concept. The points equidistant from given two points lie on a line, and the points equidistant from a point and a line lie on a parabola. The Voronoi diagram, an important structure in computational geometry, can be considered as bisectors generalized to n points [3, 5].

What happens if bisection is replaced by trisection, or more? Asano et al. [2] introduced the *distance k-sector* (henceforth just *k-sector*) by extending the equidistance condition for bisection: the k-sector of two disjoint closed sets P and Q in the Euclidean space $\mathbb{R}^d$ is a series of $k-1$ nonempty sets $C_1, \dots, C_{k-1} \subseteq \mathbb{R}^d$ such that each C_i is the bisector of C_{i-1} and C_{i+1}, where we regard $C_0 = P$ and $C_k = Q$.

Consider the most basic case where P and Q are points in the plane. In this case, the trisector (3-sector) exists and is unique [2]. The existence of a 4-sector of two points is easy: let C_2 be the perpendicular bisector of P and Q, and let C_1 and C_3 be the bisecting parabolas. We do not know whether this is the unique solution. Chun et al. [4] have shown that the trisector of a point and a line exists and is unique, and thus a 6-sector of two points exists; see Figure 1. We remark that C_1 and C_5 of this 6-sector are closed curves. Reem and Reich [6] proved that a trisector of given two sets always exists in any metric space. It was conjectured that a k-sector exists for any k, but has not been proved even for two points in the plane.

Combining the ideas of Voronoi diagrams and the trisector, we obtain the notion of *zone diagrams* [1]. A zone diagram of nonempty sets (called *sites*) $P_1, \dots, P_n \subseteq \mathbb{R}^d$ is an n-tuple $(R_1, \dots, R_n)$ of subsets of $\mathbb{R}^d$ satisfying

$$R_i = \operatorname{dom}\Big(P_i, \bigcup_{j \neq i} R_j\Big), \qquad i = 1, \dots, n, \qquad (1)$$

where, for sets A, $B \subseteq \mathbb{R}^d$ (not both empty), $\operatorname{dom}(A, B) = \{x \in \mathbb{R}^d : d(x, A) \le d(x, B)\}$ is the *dominance region* of A over B, with $d(X, Y) = \inf_{x \in X, y \in Y} |x - y| \in [0, +\infty]$ denoting the Euclidean distance of sets X and Y. Figure 2 shows the zone diagram (and the Voronoi diagram) of line segments

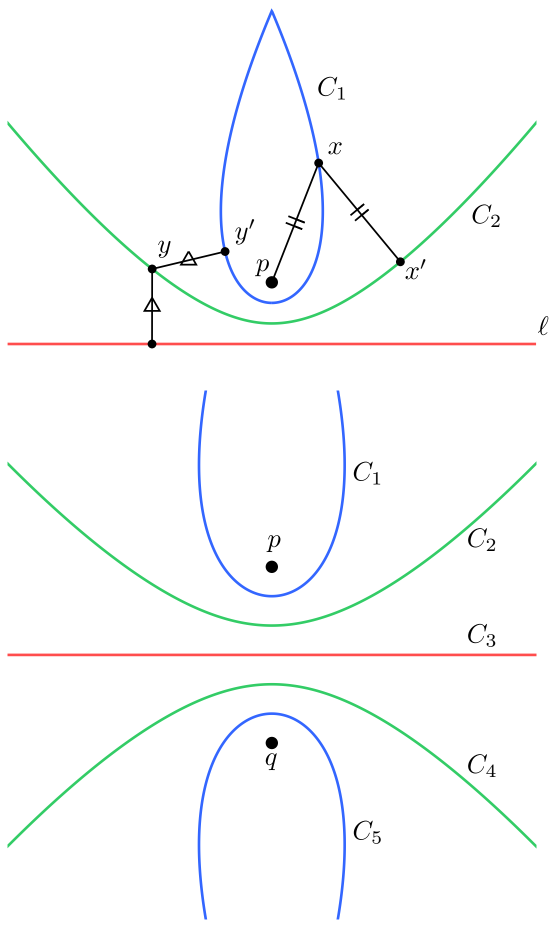


Figure 1: The trisector of line l and point p (top) and a 6-sector of points p, q (bottom).

[*]Department of Information and System Engineering, Faculty of Science and Engineering, Chuo University

[†]Department of Computer Science, University of Toronto. Supported in part by the Natural Sciences and Engineering Research Council of Canada and the Nakajima Foundation.

[‡]Department of Applied Mathematics, Charles University

[§]Graduate School of Science and Engineering, Chuo University

[¶]Graduate School of Information Sciences, Tohoku University

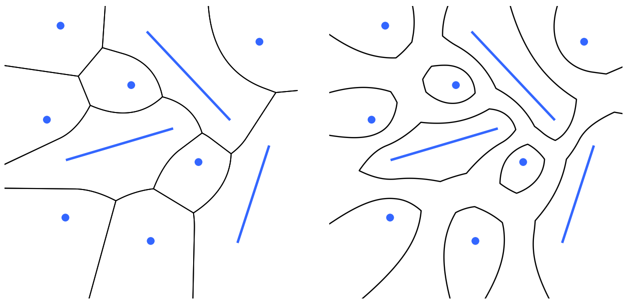


Figure 2: The Voronoi diagram (left) and the zone diagram (right) of points and line segments.

and points in the plane. A zone diagram of two disjoint closed sites gives their trisector. Asano et al. [1] proved the existence and uniqueness of the zone diagram of point sites in the plane, and conjectured that the same is true for general sites and general dimensions. Unfortunately, their proof involves case analysis specific to $\mathbb{R}^2$ that seems hard to generalize.

The concept of a zone diagram can be immediately generalized to any metric space. Reem and Reich [6] proved the existence of what they call *double zone diagrams*, which are candidates for zone diagrams (see Section 3), in an arbitrary metric space (and even in a still more general setting, which they call *m-spaces*). The existence of a zone diagram is not known in general, and there are counterexamples to uniqueness in some artificial metric spaces [6].

We settle two of the conjectures mentioned above for Euclidean space:

Theorem 1 *Let $P_1, \ldots, P_n$ be nonempty subsets of $\mathbb{R}^d$ such that $d(P_i, P_j) > 0$ for every $i \neq j$. Then the zone diagram of $(P_1, \ldots, P_n)$ exists and is unique.*

Theorem 2 *For any disjoint nonempty closed sets P and $Q \subseteq \mathbb{R}^d$, there exists a k-sector of P and Q.*

These will be proved in Sections 3 and 4.1, respectively. The *Knaster–Tarski fixed point theorem* (Section 2), used already in [6], will play an essential role.

Theorem 1 is new even for the case $n = 2$, since uniqueness was known only when the sites are points. The proof is simpler than the ones in [2, 1] even for the case of point sites in the plane.

Given a set $R \subseteq \mathbb{R}^d$, we write $\overline{R}$, ∂R and R^c for its closure, boundary and complement.

2 The Knaster–Tarski fixed point theorem

Fix $n \in \mathbb{N}$ and let $\mathcal{L}$ be the set of all ordered n-tuples of subsets of $\mathbb{R}^d$. Given $\mathbf{A} = (A_1, \ldots, A_n) \in \mathcal{L}$ and $\mathbf{B} = (B_1, \ldots, B_n) \in \mathcal{L}$, define $\mathbf{A} \leq \mathbf{B}$ if $A_i \subseteq B_i$ for each i. This gives a partial ordering on $\mathcal{L}$. A function g from $\mathcal{S} \subseteq \mathcal{L}$ to $\mathcal{S}$ is *monotone* (resp. *anti-monotone*) if $g(\mathbf{A}) \leq g(\mathbf{B})$ (resp. $g(\mathbf{B}) \leq g(\mathbf{A})$) for any $\mathbf{A} \leq \mathbf{B}$. An n-tuple $\mathbf{D} \in \mathcal{L}$ is an *upper* (resp. a *lower*) *bound* of $\mathcal{Z} \subseteq \mathcal{L}$ if $\mathbf{X} \leq \mathbf{D}$ (resp. $\mathbf{D} \leq \mathbf{X}$) for any $\mathbf{X} \in \mathcal{Z}$. A set $\mathcal{S} \subseteq \mathcal{L}$ is called a *complete lattice* if any subset $\mathcal{Z} \subseteq \mathcal{S}$ has the least upper bound $\bigvee \mathcal{Z}$ and the greatest lower bound $\bigwedge \mathcal{Z}$ in $\mathcal{S}$. Following Reem and Reich [6], we will use the Knaster–Tarski Theorem:

Theorem 3 ([7]) *If $\mathcal{S}$ is a complete lattice and $g\colon \mathcal{S} \to \mathcal{S}$ is monotone, then $\mathbf{R} = \bigwedge\{\, \mathbf{Y} \in \mathcal{S} : g(\mathbf{Y}) \leq \mathbf{Y} \,\}$ and $\mathbf{S} = \bigvee\{\, \mathbf{Y} \in \mathcal{S} : g(\mathbf{Y}) \geq \mathbf{Y} \,\}$ are fixed points of g. Moreover, $\mathbf{R} \leq \mathbf{X} \leq \mathbf{S}$ for any fixed point $\mathbf{X}$.*

In our applications, the monotonicity required in the theorem is based on the following simple observation [1, 6].

Lemma 4 *For $X \subseteq X'$, we have $\operatorname{dom}(X, B) \subseteq \operatorname{dom}(X', B)$ and $\operatorname{dom}(A, X) \supseteq \operatorname{dom}(A, X')$.*

3 Existence and uniqueness of zone diagrams

Let $\mathcal{S}$ be the complete lattice consisting of all n-tuples of subsets of $\mathbb{R}^d$. Let us fix an n-tuple $\mathbf{P} = (P_1, \ldots, P_n) \in \mathcal{S}$ of nonempty sets (the sites). Let $\mathbf{Dom}\colon \mathcal{S} \to \mathcal{S}$ be given by $\mathbf{Dom}(\mathbf{D}) = \mathbf{E}$, where $\mathbf{D} = (D_1, \ldots, D_n)$ and $\mathbf{E} = (E_1, \ldots, E_n)$ with

$$E_i = \operatorname{dom}\Big(P_i, \bigcup_{j \neq i} D_j\Big), \qquad i = 1, \ldots, n. \tag{2}$$

A zone diagram of $\mathbf{P}$ is exactly a fixed point of $\mathbf{Dom}$ [1].

The function $\mathbf{Dom}$ is anti-monotone (Lemma 4), hence $\mathbf{Dom}^2 = \mathbf{Dom} \circ \mathbf{Dom}$ is monotone. Reem and Reich [6] call a fixed point of $\mathbf{Dom}^2$ a *double zone diagram* of $\mathbf{P}$. From Theorem 3 one obtains the following:

Theorem 5 ([6, Theorem 5.5]) *The function $\mathbf{Dom}^2$ has fixed points $\mathbf{R}$ and $\mathbf{S}$ such that $\mathbf{R} = \mathbf{Dom}\,\mathbf{S}$, $\mathbf{S} = \mathbf{Dom}\,\mathbf{R}$ and $\mathbf{R} \leq \mathbf{D} \leq \mathbf{S}$ for any fixed point $\mathbf{D}$ of $\mathbf{Dom}^2$.*

Using this, we will now prove Theorem 1. We remark that the same proof works also for infinitely many sites, every two of them at distance at least ε, for some fixed $\varepsilon > 0$.

We may assume that the sites $P_1, \ldots, P_n$ are closed (since a zone diagram of their closures is also a zone diagram of the P_i).

Lemma 6 *Let $P_1, \ldots, P_n$ be closed and let ε be as above. Suppose that n-tuples $\mathbf{D}$ and $\mathbf{E}$ of subsets of $\mathbb{R}^d$ satisfy $\mathbf{D} = \mathbf{Dom}\,\mathbf{E}$ and $\mathbf{E} = \mathbf{Dom}\,\mathbf{D}$. Let $i \in \{1, \ldots, n\}$ and suppose that p is the closest point to $a \in D_i$ in P_i. Then the convex hull of $K \cup \{a\}$ is contained in D_i, where K is the closed ball with centre p and radius $\varepsilon/4$. (Proof omitted.)*

To prove Theorem 1, it suffices to show that the fixed points $\mathbf{R} = (R_1, \dots, R_n)$ and $\mathbf{S} = (S_1, \dots, S_n)$ in Theorem 5 coincide. Suppose, for contradiction, that there are i_0 and $b_0 \in S_{i_0} \setminus R_{i_0}$. We define index i_t and points b_t, p_t, a_t for each $t \in \mathbb{N}$ inductively as follows (Figure 3). Suppose that i_t and b_t have been defined. Let p_t be one of the closest points in P_{i_t} to b_t, and let a_t be the unique point (by Lemma 6) where the line segment $b_t p_t$ meets ∂R_{i_t}. Since $a_t \in \partial R_{i_t}$ and $\mathbf{R} = \mathbf{Dom}\,\mathbf{S}$, there are $i_{t+1} \neq i_t$ and $b_{t+1} \in S_{i_{t+1}}$ with $|a_t - b_{t+1}| = |a_t - p_t|$; choose any such.

For each $t \in \mathbb{N}$, let $r_t = |a_t - p_t|$, $s_t = |b_t - p_t|$ and $\theta_t = \angle p_t a_t b_{t+1}$. Since $|a_{t+1} - b_t| \geq s_t$ by $a_{t+1} \in R_{i_{t+1}}$, $b_t \in S_{i_t}$ and $\mathbf{S} = \mathbf{Dom}\,\mathbf{R}$, we have

$$\begin{aligned} s_{t+1} - r_{t+1} &= |a_{t+1} - b_{t+1}| \geq |a_{t+1} - b_t| - |b_t - b_{t+1}| \\ &= |a_{t+1} - b_t| - \sqrt{\begin{aligned}&|a_t - b_t|^2 + |a_t - b_{t+1}|^2 \\ &\quad + 2|a_t - b_t| \cdot |a_t - b_{t+1}| \cos\theta_t\end{aligned}} \\ &\geq s_t - \sqrt{(s_t - r_t)^2 + {r_t}^2 + 2(s_t - r_t) r_t \cos\theta_t} \\ &\geq \frac{r_t(s_t - r_t)}{s_t}(1 - \cos\theta_t). \end{aligned} \tag{3}$$

Let h be the distance from b_{t+1} to the convex hull of Lemma 6 with $i = i_t$ and $a = a_t$. Since $\mathbf{S} = \mathbf{Dom}\,\mathbf{R}$ and $b_{t+1} \in S_{i_{t+1}}$, we have

$$\begin{aligned} s_{t+1} \leq h &= r_t \sin\min\left\{\frac{\pi}{2}, \theta_t - \arcsin\frac{\varepsilon}{4r_t}\right\} \\ &\leq r_t \sin\min\left\{\frac{\pi}{2}, \theta_t - \gamma\right\}, \end{aligned} \tag{4}$$

where $\gamma = \arcsin(\varepsilon/4r_0) \leq \theta_t$. The last inequality is because the r_t are decreasing.

Let $\lambda \in (0, 1)$ be small enough that $(1 - \cos\gamma)^\lambda > \cos(\gamma/2)$. By (3) and (4), we have

$$\frac{(s_{t+1} - r_{t+1})^\lambda}{s_{t+1}} \geq \frac{(s_t - r_t)^\lambda}{r_t^{1-\lambda} s_t^\lambda} B \geq \frac{(s_t - r_t)^\lambda}{s_t} B, \tag{5}$$

where

$$\begin{aligned} B &= \frac{(1 - \cos\theta_t)^\lambda}{\sin\min\{\pi/2, \theta_t - \gamma\}} \\ &\geq \begin{cases} \dfrac{(1 - \cos\gamma)^\lambda}{\cos(\gamma/2)} & \text{if } \theta_t < \dfrac{\pi}{2} + \dfrac{\gamma}{2}, \\ \dfrac{\big(1 + \sin(\gamma/2)\big)^\lambda}{\sin(\pi/2)} & \text{otherwise} \end{cases} \\ &\geq \min\left\{\frac{(1 - \cos\gamma)^\lambda}{\cos(\gamma/2)}, \big(1 + \sin(\gamma/2)\big)^\lambda\right\} > 1. \end{aligned} \tag{6}$$

Thus, B is bounded from below by a constant exceeding 1 independently of t. Therefore, (5) implies that $(s_t - r_t)^\lambda / s_t$ is unbounded as $t \in \infty$, contradicting $(s_t - r_t)^\lambda / s_t \leq s_t^{-1+\lambda} \leq (\varepsilon/4)^{-1+\lambda}$. We have proved Theorem 1.

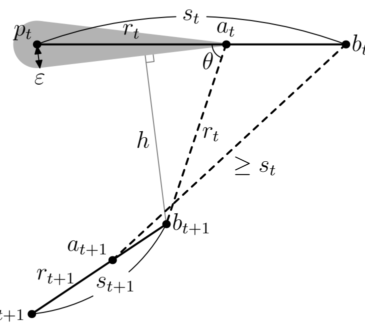


Figure 3: The shaded region is in R_{i_t} by Lemma 6.

4 Distance k-sectors

The *bisector* of nonempty subsets A and B of $\mathbb{R}^d$ is defined by

$$\operatorname{bisect}(A, B) = \{\, z \in \mathbb{R}^d : d(z, A) = d(z, B) \,\}. \tag{7}$$

It is not hard to see that

$$\operatorname{bisect}(A, B) = \partial \operatorname{dom}(A, B) = \partial \operatorname{dom}(B, A) \tag{8}$$

if the closures of A and B are disjoint. Let P and Q be disjoint nonempty closed subsets of $\mathbb{R}^d$. A k-sector of P and Q is a sequence $(C_1, \dots, C_{k-1})$ of nonempty subsets of $\mathbb{R}^d$ satisfying

$$C_i = \operatorname{bisect}(C_{i-1}, C_{i+1}), \qquad i = 1, \dots, k-1, \tag{9}$$

where $C_0 = P$ and $C_k = Q$.

4.1 Existence

Similarly to Section 2, let $\mathcal{L}$ be the complete lattice of all $(k-1)$-tuples of subsets of $\mathbb{R}^d$ (ordered by componentwise inclusion). For $\mathbf{D} = (D_1, \dots D_{k-1}) \in \mathcal{L}$ we define $F(\mathbf{D}) = (E_1, \dots E_{k-1})$ by

$$E_i = \operatorname{dom}(D_{i-1} \cup P, D_{i+1}^c \cup Q), \quad i = 1, \dots, k-1, \tag{10}$$

where $D_0 = P$ and $D_k = Q^c$. Since F is monotone by Lemma 4, it has a fixed point $(R_1, \dots, R_{k-1})$ by Theorem 3. The following lemma says that a fixed point consists of a hierarchy of sets $P = R_0 \subseteq R_1 \subseteq \cdots \subseteq R_n = Q$ with separated boundaries.

Lemma 7 *If $(R_1, \dots, R_{k-1})$ is a fixed point of the function F above, then $R_i \cap \overline{R_j^c} = \emptyset$ for each i and j with $0 \leq i < j \leq k$. (Proof omitted.)*

We are now ready to prove Theorem 2. Let $(R_1, \dots, R_{k-1})$ be a fixed point as above. Let $C_i = \partial R_i$ for each i. Then (9) is proved as follows:

$$\begin{aligned} C_i &= \partial \operatorname{dom}(R_{i-1} \cup P, R_{i+1}^c \cup Q) \\ &= \partial \operatorname{dom}(R_{i-1}, R_{i+1}^c) \\ &= \operatorname{bisect}(R_{i-1}, R_{i+1}^c) = \operatorname{bisect}(C_{i-1}, C_{i+1}). \end{aligned} \tag{11}$$

The third equality is by (8) using Lemma 7. The last equality is because $d(a, X) = d(a, \partial X)$ for $a \notin X$.

4.2 The least and the greatest solutions

We conjecture that the k-sector of two disjoint closed sites $P, Q \subset \mathbb{R}^d$ is always unique. Here we present some supporting evidence.

Let $\mathcal{L}$ and F be as above. Let $\bot = (\emptyset, \dots, \emptyset)$ be the least element of $\mathcal{L}$. For $\mathbf{D} \in \mathcal{L}$ satisfying $\mathbf{D} \le F(\mathbf{D})$, denote by $F^\infty(\mathbf{D})$ the componentwise closure of $\bigvee\{ F^n(\mathbf{D}) : n \in \mathbb{N} \}$.

Lemma 8 *$F^\infty(\bot)$ is a fixed point of F. (Proof omitted.)*

In fact, $F^\infty(\bot)$ is the least fixed point, because by monotonicity, $F^\infty(\bot) \le F^\infty(\mathbf{D}) = \mathbf{D}$ for any fixed point $\mathbf{D}$. We can prove similarly that $F^\infty(\top) = \bigwedge\{ F^n(\top) : n \in \mathbb{N} \}$ is the greatest fixed point, where $\top = (\mathbb{R}^d, \dots, \mathbb{R}^d)$. This gives, in this particular setting, a "constructive" description of the smallest and largest fixed points from Theorem 3. Thus the fixed point of F is unique if $F^\infty(\bot) = F^\infty(\top)$.

Using this, we computed, for point sites $P = \{(0, 1)\}$ and $Q = \{(0, -1)\}$, a lower bound of $F^\infty(\bot)$ for each $k \le 11$ by iteratively applying F on the pixel grid, each time underestimating the sets. We did this experiment with several different pixel sizes, and it seemed that the picture always eventually becomes symmetric with respect to the x-axis up to a few pixels, except near the edges of the grid. Thus, even a lower bound of the least fixed point $F^\infty(\bot)$ is nearly symmetric; this implies that $F^\infty(\bot)$ and $F^\infty(\top)$ (which must be symmetric to each other) are very close, although we do not have a proof that they coincide.

5 Layered zone diagrams

Zone diagrams and k-sectors were defined by simple equations involving the dom operator. We can consider various similar systems of equations. For example, given sites $P_1, \dots, P_n$, consider the equations

$$R_i = \mathrm{dom}\Big(T_i, \bigcup_{j\neq i} T_j\Big), \qquad T_i = \mathrm{dom}(P_i, R_i^c) \qquad (12)$$

for $i = 1, \dots, n$. These equations specify the relation that the layered zones $R_i \supseteq T_i \supseteq P_i$ should satisfy. They imply that, between sets P_i and P_j that are located close to each other, there are four regions bounded by their 4-sector. Thus, a solution to (12) can be regarded as a generalization to $k = 4$ of Voronoi diagrams (which correspond to $k = 2$) and of zone diagrams (which correspond to $k = 3$). We thus call a solution to (12) a *layered zone diagram* of the P_i (with four layers in this case).

The whole space is divided into $R_1, \dots, R_n$. One might perhaps expect that R_i is the Voronoi region of P_i among the sites (which is true for the case of two point sites, where we deal with a 4-sector), but it is not: For $P_1 = \{(-1, 0)\}$, $P_2 = \{(0, 0)\}$, $P_3 = \{(1, 0)\}$, the Voronoi region of P_2 is a vertical strip, but it can be proved that R_2 is bounded (Figure 4).

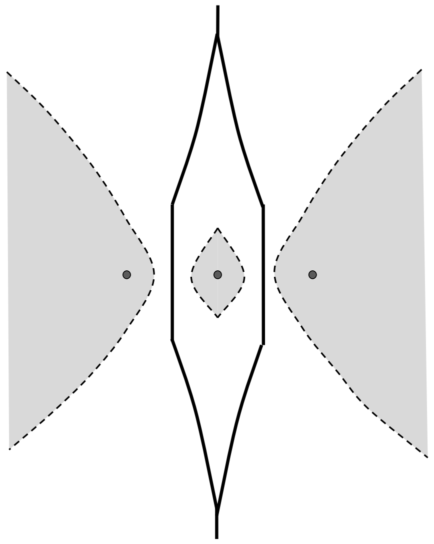


Figure 4: A layered zone diagram on three point sites. The shaded regions are the T_i.

Currently we do not have a proof of existence of such layered zone diagrams. But it is easy to show, using Theorem 3 as usual, that an appropriate "double layered zone diagram" exists, and this is a natural candidate for a layered zone diagram.

References

[1] T. Asano, J. Matoušek, and T. Tokuyama. Zone diagrams: Existence, uniqueness, and algorithmic challenge. *SIAM Journal on Computing*, 37(4):1182–1198, 2007. Short version in *Proc. SODA 2007*, 756–765.

[2] ———. The distance trisector curve. *Advances in Mathematics*, 212(1):338–360, 2007. Short version in *Proc. STOC 2006*, 336–343.

[3] F. Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computuing Surveys*, 23(3):345–405, 1991.

[4] J. Chun, Y. Okada, and T. Tokuyama. Distance trisector of segments and zone diagram of segments in a plane. In *Proc. ISVD 2007*, 66–73.

[5] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Probability and Statistics. Wiley, second edition, 2000.

[6] D. Reem and S. Reich. Zone and double zone diagrams in abstract spaces. arXiv:0708.2668v1, 2007.

[7] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

Finding a Minimum Stretch of a Function*

Kevin Buchin† Maike Buchin* Marc van Kreveld* Jun Luo‡

Abstract

Given a piecewise monotone function $f : \mathbb{R} \to \mathbb{R}$ and a real value $T_{\min}$, we develop an algorithm that finds an interval of length at least $T_{\min}$ for which the average value of f is minimized. The run-time of the algorithm is linear in the number of monotone pieces of f if certain operations are available in constant time for f. We use this algorithm to solve a basic problem arising in the analysis of trajectories: Finding the most similar subtrajectories of two given trajectories, provided that the duration is at least $T_{\min}$. Since the precise solution requires complex operations, we also give a simple $(1+\varepsilon)$-approximation algorithm in which these operations are not needed.

1 Introduction

Where does a function f have its extremes? If we look at f at a larger scale, a more useful answer to this question than the singular extrema of f may be high and low "plateaus" of f. Therefore, we consider the problem of finding an interval of the domain of f for which the average of f is minimum. The interval should have at least a given length, otherwise its length would always be zero. We develop an algorithm for this problem for functions which are piecewise monotone. The run-time of the algorithm is linear in the number of monotone pieces of f. The straightforward algorithm of iterating over all possible start and end pieces, using some precomputed values, and optimizing, would have quadratic run-time.

Our study of this problem is motivated by geometric problems occurring in geographic data analysis, in particular, the problem of finding similar subtrajectories of moving objects [6]. Given two trajectories, we wish to determine a time interval of at least a certain length such that the trajectories are close during that time interval. By "close" we mean that the average distance during the time interval is as small as possible. This application, however, is an instance of the more general problem we are solving in this paper. Another geographic application we are interested in is the following. Assume a moving object measuring some quantity while it moves, for instance, the height. We want to find high (or low) plateaus of this quantity. There are more ways to measure the similarity of trajectories, see for example the references given in [6].

The discrete version of the minimum stretch problem occurs in biological sequences alignment and has been studied there. Several similar linear-time algorithms have been given [2, 4, 5], which provide the basis of the ideas used in our algorithm. Our algorithm is considerably more complex, however, due to allowing any start and end point of the interval, and allowing any type of piecewise monotone function. For the discrete sequence version there is also a geometric algorithm using very different ideas [1].

2 Algorithm

In this section, we develop an algorithm that minimizes the average height over intervals of a piecewise monotone function f, where the interval has a non-fixed duration $T \geq T_{\min}$. The run-time of the algorithm is linear in the number of pieces of f.

Fixed length. Before we give the algorithm, we consider a simple version of the problem where the length of the interval is fixed to be $\hat{T}$. For this problem, a trivial linear-time algorithm exists by scanning the function and maintaining its average. The solution with $\hat{T} = T_{\min}$ is a 2-approximation for the problem with non-fixed length, assuming f is nonnegative. To see this, note that always an optimal length T with $T_{\min} \leq T < 2T_{\min}$ exists (for larger T, split in the middle and choose a half with smaller or equal average). If the interval I_{opt}, with smallest average, has a duration T' with $T_{\min} \leq T' \leq 2T_{\min}$, then the dissimilarity for any subinterval of length $T_{\min}$ is larger by at most a factor $T'/T_{\min} \leq 2$.

Furthermore, the factor two can be obtained in the limit, as demonstrated in Figure 1: the total duration is $2T_{\min} - \varepsilon$ and the distance between the trajectories is 0 except for a duration of ε in the middle (for illustration purposes, they are shown with a small vertical offset). In this example, fixed and non-fixed duration differ by a factor of $(2T_{\min} - \varepsilon)/T_{\min}$ which is $2 - \varepsilon$ for $T_{\min} = 1$.

*This research has been funded by the Netherlands' Organisation for Scientific Research (NWO) under BRICKS/FOCUS project no. 642.065.503 and by the German Research Foundation (DFG) under grant BU 2419/1-1.

†Department of Information and Computing Sciences, Utrecht University, 3584CH Utrecht, The Netherlands, {buchin, maike, marc}@cs.uu.nl

‡Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, China, jun.luo@sub.siat.ac.cn

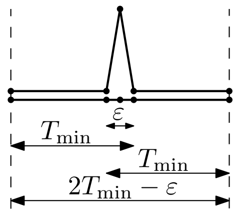


Figure 1: Worst-case ratio example.

If we run the fixed duration algorithm with durations $T_{\min}$, $(1+\varepsilon)\cdot T_{\min}$, $(1+2\varepsilon)\cdot T_{\min}$, $\ldots$, $2\cdot T_{\min}$, and take the overall optimum, then we have a $(1+\varepsilon)$-approximation algorithm for the general problem (assuming f is nonnegative) that runs in $O(n/\varepsilon)$ time.

Concept. We first illustrate the idea of the algorithm. We solve the problem by a sweep over the domain of f. At any time t_{end} we include at least a window of the minimum length $T_{\min}$ to the left of t_{end}. Additionally it may lower the average to include a part even further to the left. To decide efficiently how much of this part to include, we decompose and store this part in a data structure.

Let t_{end} be the end of some time interval, and we are interested in a minimum average value of f over an interval of length at least $T_{\min}$ that ends at t_{end}. Let $t_{\text{pre}} = t_{\text{end}} - T_{\min}$ be the last moment where the interval can start. We may want to start the interval earlier, to lower the average value of f over the chosen interval (see Figure 2). We need a careful analysis of the situation before t_{pre} to decide what the optimal starting time is for an interval that ends at t_{end}. We will store this situation in a data structure that will be updated when t_{end} and t_{pre} move simultaneously further in time. There will be events when t_{end} or t_{pre} pass a break point of the function f, but there will also be events if the situation before t_{pre} changes in a structural way.

For any interval $I = [t, t']$ we define $\bar{f}(t', t'')$ as

$$\bar{f}(I) := \bar{f}(t', t'') = \frac{\int_{t'}^{t''} f(t)dt}{t'' - t'} .$$

If $t' = t''$, we let $\bar{f}(t', t'') = \bar{f}(t', t') := f(t')$. We also define $\bar{f}(t') := \bar{f}(t', t_{\text{end}})$, which is valid if t_{end} is fixed.

For description purposes, we fix t_{end} and therefore t_{pre} for the moment. Then interval $[t_{\text{pre}}, t_{\text{end}}]$ gives an average of

$$\bar{f}(t_{\text{pre}}) = \bar{f}(t_{\text{pre}}, t_{\text{end}}) = \frac{\int_{t_{\text{pre}}}^{t_{\text{end}}} f(t)dt}{t_{\text{end}} - t_{\text{pre}}} .$$

It is clear that if the function value of f is smaller than $\bar{f}(t_{\text{pre}})$ just before t_{pre}, then extending the interval to a starting time before t_{pre} will give a lower average $\bar{f}(.)$. Even if the function value of f just before t_{pre} is greater than $\bar{f}(t_{\text{pre}})$, then extending the interval to a starting time (sufficiently far) before t_{pre} may still give a lower average. In Figure 2 we observe that the optimal starting time $t_{\text{opt}} \le t_{\text{pre}}$, given t_{end} as the end of the interval, is such that $\bar{f}(t_{\text{opt}}) = f(t_{\text{opt}})$, or $t_{\text{opt}} = t_{\text{pre}}$.

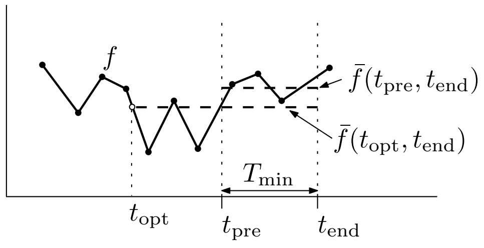

f
$\bar{f}(t_{\text{pre}}, t_{\text{end}})$
$\bar{f}(t_{\text{opt}}, t_{\text{end}})$
$T_{\min}$
t_{opt}
t_{pre}
t_{end}

Figure 2: Averages of f over $[t_{\text{pre}}, t_{\text{end}}]$ and $[t_{\text{opt}}, t_{\text{end}}]$.

If t_{end} is fixed, then the value of $\bar{f}(t_{\text{pre}})$ only determines where t_{opt} is. The time t_{opt} is monotonically decreasing in the value of $\bar{f}(t_{\text{pre}})$ (if the average of f over $[t_{\text{pre}}, t_{\text{end}}]$ were larger, we may have to go further back with t_{opt}, but never forward).

Assumptions. To compute the minimum stretch in linear time we need to assume that the following operations can be performed in constant time:

1. Evaluate the integral of f over a monotone piece.
2. Solve equations of the form $F(a, s) = as + b$, where $F(a, s) = \int_a^s f(t)dt$.
3. Find a stretch of minimum average value, if the monotone pieces for the left and the right endpoint of the stretch are given and the integral of f for the intervals in between has been evaluated.

For simplicity, we will assume that f is continuous. We can extend the ideas to handle non-continuous functions, but the definitions and description of the method become more tedious.

Data structure. Let f be a piecewise monotone function with break points $t_1, \ldots, t_n$, that is, f is monotone in between each pair t_i and t_{i+1} for $1 \le i < n$. At all times our data structure consists of the interval $I_0 = [t_{\text{pre}}, t_{\text{end}}]$ and a set of intervals $I_1, \ldots, I_m$, where $I_i = [s_i, s_{i-1}]$, for $i = 1, \ldots, m$, $m \ge 0$, and $s_m < s_{m-1} < \ldots < s_1 < s_0 = t_{\text{pre}}$. To define $s_1, \ldots, s_m$ and m, we first define a function $l(s)$ which, intuitively, tells how far to the left we can always extend an interval if we extend at least a fraction to the left of s, and still lower the average $\bar{f}$. We define l on the domain of f by

$$l(s) := \min(s' \le s \mid \forall\, 0 \le t' \le s : \bar{f}(s', s) \le \bar{f}(t, s)\ )$$

Note that if for no $s' < s$ we have $\bar{f}(s', s) < f(s)$, then $l(s) = s$. This can only happen if f is a decreasing function at s (to its left).

We can now define the s_i, $1 \le i \le m$, by

$$s_i := \begin{cases} l(s_{i-1}) & \text{if } l(s_{i-1}) < s_{i-1} \\ \max(\{t_j < s_{i-1} \mid 1 \le j \le n\} \\ \quad \cup \{s' < s_{i-1} \mid l(s') < s'\}) & \text{else.} \end{cases}$$

Thus, if $l(s_{i-1}) = s_{i-1}$, then we set s_i either to the next breakpoint t_j of f left of s_{i-1}, or to the largest $s' < s_{i-1}$ such that $l(s') < s'$. If $s_i = l(s_{i-1})$, then f must be decreasing just left of s_i.

There are two types of intervals in $I_1, \dots, I_m$: those where $s_i = l(s_{i-1})$ and those where $s_i < l(s_{i-1})$. We will call the first type of intervals *complete* and the other type *decreasing*. These intervals have the following properties:

1. If I_i is complete and $i > 1$, then for all $s' \in [s_i, s_{i-1}]$ we have $f(s_{i-1}) = f(s_i) = \bar{f}(I_i)$.
2. If I_i is complete, then for all $s' \in (s_i, s_{i-1})$ we have $\bar{f}(s_i) < \bar{f}(s')$, and also I_{i+1} is decreasing if $i \ge 1$.
3. $\bar{f}(I_i) < \bar{f}(I_j)$ if and only if $i < j$.

Note that the first property does not hold for I_1 because it is not preceded by a decreasing interval. The last property states that the average gets higher to the left. Any complete interval contains a break point of f, and consecutive decreasing intervals are separated by a break point of f. Together with the second property, this implies $m = O(n)$.

The integer m, representing the last interval to the left that we need to consider, depends on the average height of f over intervals in the data structure. We will not need to consider intervals at the left end of our data structure if their (partial) inclusion would increase the average height. It follows that the last interval I_m that we need is a decreasing interval. Also, we do not need intervals further to the left if their inclusion would result in an average height which is larger than a previously found average height. Hence, we will have:

$$f(s_m) \ge \min_{t'+T_{\min} \le t \le t_{\text{end}}} \bar{f}(t', t) \ge f(s_{m-1}) \ .$$

Our data structure maintains the sequence of break points $t_{\text{end}}, s_0, s_1, \dots, s_m$, the pieces of f that contains each, and the sequence $F(I_0), \dots, F(I_m)$, where $F(I_i) = \int_{s_{i-1}}^{s_i} f(t)dt$. The sequences can simply be stored in a list or an array. During the algorithm, we only change information at the ends of the sequences. We also maintain $F(I_{m-1} \cup \dots \cup I_1)$.

Algorithm. We can find the interval with minimum average height as follows. We scan with the interval $[t_{\text{pre}}, t_{\text{end}}]$ from start to end along the domain of the function f, and maintain the information we just described. Most of this information can only change at certain discrete event points that we handle during the scan. The positions of t_{end}, s_0, and possibly s_1 change continuously, but we will use the maintained information and their notation as it was valid at the last event. We use t'_{end}, s'_0, s'_1, I'_0, etc., to denote the corresponding values that are valid at the next event, and $\tilde{t}_{\text{end}}$, $\tilde{t}_{\text{pre}} = \tilde{s}_0$, $\tilde{s}_1$, $\tilde{I}_0$, etc., to denote values in between events t_{end} and t'_{end}.

In between two consecutive event points $t_{\text{end}} \le \tilde{t} \le t'_{\text{end}}$, we need to minimize $\bar{f}(t, \tilde{t})$ over the choices of t and $\tilde{t}$ with $t \le \tilde{t} - T_{\min}$, where we know on which pieces of f the interval endpoints t and $\tilde{t}$ lie. To minimize $\bar{f}(t, \tilde{t})$, we find the expressions for $F(t, \tilde{s}_{m-1}) = F(t, s_{m-1})$ and $F(\tilde{I}_{m-1} \cup \dots \cup \tilde{I}_0) = F(I_{m-1} \cup \dots \cup I_3 \cup \tilde{I}_2 \cup \tilde{I}_1 \cup \tilde{I}_0)$ in the unknowns t and $\tilde{t}$, and minimize. Since $t \in I_m$, and I_m is a decreasing interval, we have one piece of f over I_m. Hence, the expression for $F(t, s_{m-1})$ is easy to obtain in constant time. Furthermore, we maintained $F(I_{m-1} \cup \dots \cup I_1)$ and the $F(I_i)$ at the previous event point t_{end}, and $\tilde{t}$ does not pass any vertex of f before the next event, so we can derive the expression $F(\tilde{I}_{m-1} \cup \dots \cup \tilde{I}_0)$ in constant time as well. If $m = 0$, we simply take the expression $\bar{f}(\tilde{t} - T_{\min}, \tilde{t})$. By the third assumption, we can minimize such expressions in constant time. Summarizing, we can find the optimal interval between two consecutive events in constant time.

It remains to describe how we update the data structure in constant time. Instead of precomputing all event points, we will compute them dynamically.

Event points. Recall that t_{end} denotes the time of the previous event and t'_{end} denotes the time of the next event. We have four types of events.

1. $\tilde{I}_0$ moves to the next break point of f, that is, either $s'_0 = t_i$ or $t'_{\text{end}} = t_i$ for $1 \le i \le n$. If $s'_0 = t_i$ and I_1 is decreasing, then we create a new interval I'_1 that may be decreasing or complete.
2. I_1 is complete, and $\bar{f}(\tilde{I}_1)$ increases until $\tilde{I}_2$ disappears. If I_3 is decreasing, then $I'_1 = [s_2, s'_0]$; otherwise, $I'_1 = [s_3, s'_0]$ (two adjacent complete intervals merge immediately).
3. I_1 is complete, $\bar{f}(\tilde{I}_1)$ increases and $f(\tilde{s}_0)$ decreases until $\bar{f}(\tilde{I}_1) = f(\tilde{s}_0)$. We create a new decreasing interval I'_1.
4. The leftmost interval becomes irrelevant because its average height is too large, that is, $\bar{f}(s_{m-1}) \ge \min_{t \le t'_{\text{end}} - T_{\min}} \bar{f}(t, t'_{\text{end}})$. Then we discard I_m. If I_{m-1} is complete, we discard it as well.

Note that instead of stopping at events of type 4, we can also check if they happened at the next event of type 1, 2, or 3.

Computing the event points. The event points of type 1 are the break points of f, and they are known beforehand. We cannot precompute the event points of types 2, 3, and 4, but we can compute the next such event point if it is before the next type 1 event. The event points of types 2, 3, and 4 are detected as follows. Let t_{end} be the most recent event point, and let $s_0, s_1, \ldots$ be the interval endpoints with respect to t_{end}. Let t'_{end} be the next event point of type 1. An event point t of type 2 occurs for $t_{\text{end}} < t < t'_{\text{end}}$ if $f(s_2) = \bar{f}(s_2, t - T_{\min})$. To detect this, we observe

$$\bar{f}(s_2, t-T_{\min}) = \frac{F(s_2,s_1) + F(s_1,s_0) + F(s_0, t - T_{\min})}{t - T_{\min} - s_2}$$

and make an expression in t. This takes constant time using the values $F(I_2)$, $F(I_1)$, and $F(I_0)$. Then we find t by setting it equal to $f(s_2)$ (using the second assumption). Events of type 3 are detected in a similar manner.

An event point t of type 4 occurs for $t_{\text{end}} < t < t'_{\text{end}}$ if $\bar{f}(s_{m-1}, t) = f(s_{m-1})$. To detect this we solve

$$\bar{f}(s_{m-1}, t) = \frac{F(s_{m-1}, t_{\text{end}}) + F(t_{\text{end}}, t)}{t - s_{m-1}} = f(s_{m-1})$$

which we can compute in constant time as before.

Updating the data structure. At all types of event points we update the interval endpoints t_{end}, s_0, and s_1 in constant time. At an event of type 2 we discard s_1 and possibly s_2. At an event of type 4 we discard I_m and s_m, and if I_{m-1} is complete, we discard it and s_{m-1} as well. In all cases we update $F(I_0)$, $F(I_1)$, $F(I_2)$, and $F(I_{m-1} \cup \cdots \cup I_1)$, and the pieces of f that contain t_{end}, s_0, s_1, and s_m. Each of these updates can be done in constant time.

Correctness and run-time. The optimal solution is the minimal value $\bar{f}(t, \tilde{t})$ for $t \leq \tilde{t} - T_{\min}$. Assume $(t, \tilde{t})$ is such an optimal pair where $\tilde{t}$ is minimal such that it is the second part of an optimal pair and t is maximal such that it is the first part of an optimal pair with $\tilde{t}$. Let $t_{\text{end}} < \tilde{t} < t'_{\text{end}}$ be the event points left and right of $\tilde{t}$. We need to prove that $t \in I_m$.

Suppose $t < s_m$. Then t lies in an interval previously discarded. Let t'' be the right endpoint of this interval before it was discarded. Since the interval was discarded there are $\hat{t}', \hat{t}$ with $\hat{t}' \leq \hat{t} - T_{\min}$ and $\hat{t} \leq \tilde{t}$ such that $\bar{f}(t, t'') \geq \bar{f}(\hat{t}', \hat{t})$. Because of optimality of $(t, \tilde{t})$, $\bar{f}(t, t'') \geq \bar{f}(t, \tilde{t})$. Therefore, discarding the interval from t to t'' will not increase the average. This contradicts the maximality of t.

Next, suppose $s_{i-1} \geq t > s_i$ for some $i \leq m-1$. But then including the interval from s_i to t to $(t, \tilde{t})$ would decrease the average because I_m was not discarded, contradicting the optimality of $(t, \tilde{t})$.

It is not hard to see that the number of events is linear, and the running time is $O(n)$.

3 On trajectory similarity

A (time-dependent) trajectory is a continuous function from $[0,1]$ to the plane. We use the algorithm above to solve: Given two piecewise linear trajectories τ_1, τ_2 and $T_{\min} > 0$, find a time interval $[t_1, t_2]$ of length $\geq T_{\min}$ that minimizes the average distance $\int_{t_1}^{t_2} d(\tau_1(t), \tau_2(t))dt$, where d is the Euclidean distance.

On an interval on which both τ_1 and τ_2 are linear, $d(\tau_1(t), \tau_2(t)) = \sqrt{At^2 + Bt + c}$, which corresponds to a hyperbolic arc. It has no local maxima and possibly one local minimum interior to the interval. We split the intervals at such minima, so the distance between the trajectories is a piecewise monotone function and we can apply the algorithm above. It remains to see whether the operations needed for the algorithm above are available for $d(\tau_1(t), \tau_2(t))$. For a continuous function $f(t)$ (such as $d(\tau_1(t), \tau_2(t))$), a minimizing stretch $[t_1, t_2]$ with $t_2 - t_1 \geq T_{\min}$ falls in one of the following cases: $t_2 - t_1 = T_{\min}$ or $t_1 = 0$ or $t_2 = 1$ or $f(t_1) = f(t_2) = \bar{f}(t_1, t_2)$. This gives an equation in, say, t_1 such that any left endpoint of a minimizing interval is a solution to the equation. This improves a quadratic time solution in [6].

The precise solution for $d(\tau_1(t), \tau_2(t))$ requires fairly complicated operations. A $(1+\varepsilon)$-approximation can be obtained by replacing the Euclidean distance by a polyhedral distance function. We use a regular k-gon with $k = O(\sqrt{1/\varepsilon})$ vertices [3] to define it. In an interval in which both $\tau_1(t)$ and $\tau_2(t)$ are linear, this results in a piecewise linear distance function with $O(\sqrt{1/\varepsilon})$ pieces. The total run-time is then $O(n/\sqrt{\varepsilon})$.

We note that with polyhedral distance functions we can find approximate solutions for trajectory similarity with time shifts [6].

References

[1] T. Bernholt, F. Eisenbrand, and T. Hofmeister. A geometric framework for solving subsequence problems in computational biology efficiently. In *Proc. 23rd ACM Symp. on Comput. Geom.*, pages 310–318, 2007.

[2] K.-M. Chung and H.-I. Lu. An optimal algorithm for the maximum-density segment problem. *SIAM J. Comput.*, 34(2):373–387, 2005.

[3] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approximation Theory*, 10:227–236, 1974.

[4] M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu. Linear-time algorithms for computing maximum-density sequence segments with bioinformatics applications. *J. Comput. Syst. Sci.*, 70(2):128–144, 2005.

[5] Y.-L. Lin, T. Jiang, and K.-M. Chao. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci.*, 65(3):570–586, 2002.

[6] M. van Kreveld and J. Luo. The definition and computation of trajectory and subtrajectory similarity. In *Proc. 15th ACM Symp. on Advances in GIS*, pages 44–47. ACM, 2007.

Visibility Maps of Realistic Terrains have Linear Smoothed Complexity

Mark de Berg Herman Haverkort Constantinos P. Tsirogiannis*

Abstract

We study the complexity of the visibility map of so-called realistic terrains: terrains whose triangles are fat, not too steep and have roughly the same size. It is known that the complexity of the visibility map of such a terrain with n triangles is $\Theta(n^2)$ in the worst case. We prove that if the elevations of the vertices of the terrain are subject to uniform noise which is proportional to the edge lengths, then the worst-case expected (smoothed) complexity is only $\Theta(n)$. This provides an explanation why visibility maps of superlinear complexity are unlikely to be encountered in practice.

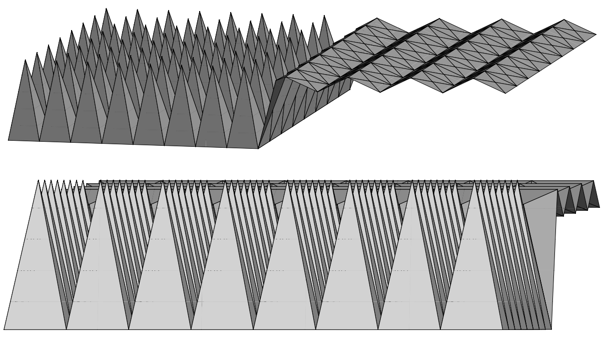


Figure 1: Two views of the same terrain defined by a regular grid. The latter view has $\Theta(n\sqrt{n})$ complexity.

1 Introduction

Motivation. A (triangulated) *terrain* is a polyhedral surface obtained by assigning elevations to the vertices of a planar triangulation. In geographic information science such terrain models are known as triangulated irregular networks, or TINs for short. Terrains can be used to model mountainous regions, as well as to approximate any scalar function defined over a planar region.

Often it is desirable to compute which parts of a terrain $\mathcal{T}$ are visible from a given viewing point p_{view}. The projections of the visible triangle parts onto a viewing plane form the *visibility map* of $\mathcal{T}$ with respect to p_{view}. Computing visibility maps is useful for visualization purposes (hidden-surface removal or shadow generation), for planning buildings under visibility constraints and for other tasks involving visibility analysis. There are several algorithms for computing visibility maps of terrains, the most efficient of which runs in time $O((n\alpha(n) + k) \log n)$ [6] where $\alpha(\cdot)$ is the inverse Ackermann function. Here n is the number of triangles in $\mathcal{T}$ and k is the *complexity* of the visibility map, which can be defined[1] as the number of vertices of the map. Each vertex of the map either corresponds to a triangle vertex, or to two edges whose projections onto the viewing plane intersect. In the worst case, $\Theta(n^2)$ pairs of edges have visible intersecting projections. In most applications a quadratic complexity would make an explicit computation of the visibility map infeasible. Fortunately such high complexity is seldom encountered. In fact, in practice it seems that the complexity of visibility maps is closer to linear. Our goal is to understand why visibility maps of terrains in practice often have low complexity.

Realistic input models. One possible approach to explain the low complexity is using a so-called *realistic input model* [5]. Here one assumes that the input has certain properties that are hopefully satisfied by inputs encountered in practice, and that rule out contrived worst-case inputs. This approach works well for many problems, and Moet et al. [9] have applied it to visibility maps of terrains. Moet et al. make the following assumptions on the terrain: the triangles of the planar triangulation defining the terrain are fat (as defined below), the triangle edges have constant length ratio, and the domain of the triangulation is a rectangle of constant aspect ratio. Unfortunately, the assumptions do not explain why visibility maps of terrains would have near-linear complexity in practice: Moet et al. showed that the worst-case complexity of the visibility map of a terrain that

*Dept. of Mathematics and Computer Science, Eindhoven University of Technology, mdberg@win.tue.nl, cs.herman@haverkort.net, ctsirogi@win.tue.nl

[1]Formally, the complexity would be defined as the total number of vertices, edges, and faces of the map. In our setting this is always linear in the number of vertices, so we restrict ourselves to this quantity.

satisfies their assumptions is $\Theta(n\sqrt{n})$. In fact, one can even assign elevations to the vertices of a triangulated grid in such a way that the triangles do not become steep while the visibility map has complexity $\Theta(n\sqrt{n})$ for certain viewing directions—see Fig. 1. To explain the linear behavior, an alternative approach is needed.

Smoothed analysis. For some problems, a small perturbation of a worst-case instance can reduce the complexity of the instance significantly. This may support the fact that such constructions are highly improbable to appear in practice. *Smoothed analysis* formalizes this idea.

Let $\mathcal{I}(n)$ be the set of all possible input instances (in our case: terrains) of size n. For an input $I \in \mathcal{I}(n)$, let $C(I)$ denote the quantity we want to analyze (the complexity of the visibility map). Furthermore, for any input $I \in \mathcal{I}(n)$ we define a neighbourhood $\mathcal{N}(I) \subset \mathcal{I}(n)$ of input instances, and we define a probability distribution over $\mathcal{N}(I)$ that indicates for every $I' \in \mathcal{N}(I)$ the probability that applying noise to the input I will result in the input I'.The smoothed (that is worst-case expected) complexity is then defined as

$$C_{smooth}(n) = \sup_{I \in \mathcal{I}(n)} E_{I' \in \mathcal{N}(I)}[C(I')]$$

where the expectation is according to the given probability distribution on $\mathcal{N}(I)$. Smoothed analysis was introduced by Spielman and Teng [10]. So far there have only been a few applications in computational geometry (see e.g. [2, 3, 4]), none of which deals with terrains. An up-to-date collection of the published works related to smoothed analysis is maintained online by Spielman [11].

Our result. We study the smoothed complexity of visibility maps of terrains under the following model:

- To each vertex's elevation noise is added that follows a uniform distribution in an interval $[-c, c]$, where c is a fixed constant fraction of the minimum edge length of the triangulation underlying the terrain.[2]

Our noise model defines for each input terrain $\mathcal{T}$ a neighbourhood $\mathcal{N}(\mathcal{T})$ consisting of those terrain instances that can be obtained by changing the elevation of each vertex by at most c, and a probability distribution on $\mathcal{N}(\mathcal{T})$. Yet, by applying a small perturbation one does not get rid of peaks that are unrealistically skinny and high, and so the smoothed visibility map complexity of arbitrary terrains is still quadratic.

Hence, we combine the power of smoothed analysis with the ideas of realistic input models. We define the following parameters of terrains:

- *Fatness:* the smallest angle of any triangle's projection onto the horizontal plane;
- *Steepness:* the largest dihedral angle between any triangle and the horizontal plane;
- *Scale factor:* the length of the longest edge divided by the length of the shortest edge of the triangulation.

We assume that the fatness ϕ, steepness θ, and scale factor σ of the unperturbed terrain are constants $\phi > 0$, $\theta < \pi/2$, $\sigma \geqslant 1$ that are independent of the number of triangles n. Some of these assumptions are also used in other papers on realistic terrains [1, 9]. In itself, our assumptions do not lead to the desired result: there are terrains satisfying these assumptions with quadratic-complexity visibility maps. For example, we can take the same construction as in Fig. 1, but with the aspect ratio of the domain being $\Theta(n)$. Our main result is that the smoothed complexity of any visibility map of a terrain satisfying the abovementioned assumptions is only $\Theta(n)$. This result holds for orthographic as well as perspective views.

2 Visibility maps resulting from perspective projection

Let $\mathcal{T}$ be a terrain with n triangles, and let E be the set of edges of $\mathcal{T}$. Let the coordinates of the vertices be specified by three coordinates x, y and z, where the z-axis is the vertical axis on which the elevation is specified. Let $\overline{\mathcal{T}}$ denote the triangulation in the xy-plane defining $\mathcal{T}$. Without loss of generality we assume that the minimum edge length in $\overline{\mathcal{T}}$ is 1. We study the smoothed complexity of the visibility map of $\mathcal{T}$ for perspective views, that is, the map as it appears in the projection on a viewing plane h_{view} as seen from a viewing point p_{view}. We assume that p_{view} is located above the terrain. Our results can be shown to hold also for orthographic views.

We denote the projection of an object o onto h_{view} by $\text{pr}(o)$. For an edge $e \in E$ we use $h_{\text{align}}(e)$ to denote the plane containing e and p_{view}. The

[2]Terrain data is usually acquired using techniques such as LIDAR. Typically, the vertical accuracy of the measurements is in the range 3–15cm while the horizontal resolution is in the range 10–100cm [8].

steepness $\theta(t)$ of a triangle t is defined as the dihedral angle of the plane containing t with the xy-plane, and the steepness $\theta(s)$ of a segment s is defined as the angle of the line containing s with the xy-plane. Observe that the steepness of a triangle is the maximum steepness of any segment contained in it. Recall that θ is the maximum steepness of any triangle in $\mathcal{T}$.

Lemma 1 *After perturbing the elevation of each terrain vertex independently by a distance of at most c, no terrain triangle is steeper than* $\theta_{\max} = \arctan(\tan(\theta) + \frac{2c}{\sin\phi})$.

As $\phi > 0$, and $\theta < \pi/2$, and c are constants, $\theta_{\max}$ is also a constant strictly smaller than $\pi/2$.

The *perceived steepness* $\theta_{\text{view}}(e)$ of an edge e is the steepness of $\text{pr}(e)$ in the plane h_{view}. In other words, $\theta_{\text{view}}(e)$ is the smallest angle between the line containing $\text{pr}(e)$ and a horizontal line on h_{view}. Even though $\theta(e) \leqslant \theta_{\max}$ by Lemma 1, an edge that is almost horizontal can appear vertical when projected onto h_{view}. We say that an edge e *appears steep* when $\theta_{\text{view}}(e)$ is not a single point and $\theta_{\text{view}}(e) > \theta_{\max}$, otherwise e *appears flat.*

We say that an edge e lies *in front of* an edge e', if there is a ray from p_{view} that, in the projection on the xy-plane, hits e before hitting e'.

A *silhouette edge* is an edge e such that the two triangles of $\mathcal{T}$ that share e are on the same side of $h_{\text{align}}(e)$. Thus one can see past e on the other side of $h_{align}(e)$. This means one of the incident triangles is front-facing while the other is back-facing. We say that two edges e and e' *create a visible intersection* if $\text{pr}(e) \cap \text{pr}(e')$ is a vertex of the visibility map. For this to be possible, the edge hit first by a ray from p_{view}—say this edge is e—must be a silhouette edge.

We observed above that even though the terrain edges are not steeper than $\theta_{\max}$, they can still appear steep on h_{view}. Yet, we can show that this cannot happen for silhouette edges.

Lemma 2 *Let* $\mathcal{T}$ *be a terrain whose triangles have steepness at most* $\theta_{\max}$*. Then the perceived steepness (on any viewing plane) of any silhouette edge of* $\mathcal{T}$ *is at most* $\theta_{\max}$.

Counting intersections. We will charge each visible intersection to the edge furthest from the viewer (of the two edges creating it). We denote with $K(e)$ the number of visible intersections an edge e creates with edges in front of it.

Suppose that we have already perturbed the edges in front of e, and we wish to analyze the effect of perturbing e. Let $E_{\text{fr}}(e)$ be the set of silhouette edges lying in front of e in the projection onto the xy-plane, excluding the edges sharing a vertex with e; the latter cannot create a visible intersection with e. Then the visible intersections charged to e are intersections of $\text{pr}(e)$ with the upper envelope of $\{\text{pr}(e') : e' \in E_{\text{fr}}(e)\}$. Consider a fragment f of an edge $e' \in E_{\text{fr}}(e)$ that appears on this upper envelope. We wish to bound the probability that after perturbation of e, $pr(e)$ intersects $pr(f)$.

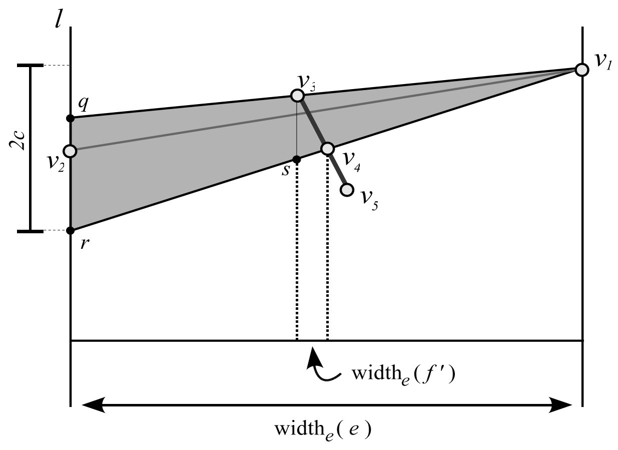


Figure 2: The shaded cone shows all the possible positions of e that may give an intersection between e and f for a fixed position of v_1.

Define $\text{span}_e(o)$ to be the projection of an object o onto a horizontal line in the vertical plane containing e, and define $\text{width}_e(o)$ to be the length of $\text{span}_e(o)$.

Lemma 3 *Consider a viewing plane* h_{view} *that is vertical and parallel to e. Let f be a fragment in front of e such that* $\text{span}_e(f) \subset \text{span}_e(e)$ *and* $\text{width}_e(f) \leqslant \text{width}_e(e)/3$*. Suppose we independently perturb the elevations of the vertices of e, with perturbations chosen uniformly at random from* $[-c, c]$*. Then the probability that e creates a visible intersection with f is at most*

$$\frac{3\,\text{width}_e(f) \cdot \tan\theta_{\max}}{c}$$

Proof. Assume w.l.o.g that the projection of e on the xy-plane is parallel to the x-axis. Let v_1 and v_2 be the vertices of e such that v_2 is the vertex closest to f in the projection onto the x-axis, with ties broken arbitrarily. Since $\text{width}_e(f) \leqslant \text{width}_e(e)/3$, the distance from v_1's projection to $\text{span}_e(f)$ is at least $\text{width}_e(e)/3$.

Let ℓ be the vertical line on h_{view} through $\text{pr}(v_2)$ and consider a fixed position of v_1. The set of all possible positions of the perturbed v_2 that induce an intersection between $pr(f)$ and $pr(e)$ is a segment $\overline{qr}$ on ℓ; —see Fig. 2. The probability that $pr(e)$ intersects $pr(f)$ is at most $|qr|/(2c)$.

The triangle $\triangle(v_1qr)$ may not contain f completely: there may be parts of f where e cannot create an intersection (for this position of v_1), because v_2 could not be perturbed that far. Let $f' = \overline{v_3v_4}$ be the part of f inside $\triangle(v_1qr)$, with v_3 being the endpoint closest to ℓ. Let s be the point such that $\triangle(v_1v_3s)$ and $\triangle(v_1qr)$ are similar triangles. $\theta_{\text{view}}(\overline{rv_1}) \leqslant \theta_{\max}$ since e is projected onto a vertical viewing plane parallel to e; f' has steepness at most $\theta_{\max}$ by Lemma 2, so:

$$|v_3s| \leqslant 2\,\text{width}_e(f') \cdot \tan\theta_{\max}.$$

Moreover, we have $|v_1v_3| \geqslant |v_1q|/3$. Now

$$|qr| \;=\; |v_3s| \cdot \frac{|v_1q|}{|v_1v_3|} \;\leqslant\; 6\,\text{width}_e(f) \cdot \tan\theta_{\max}$$

Hence, the probability of intersection between e and f for any fixed position of v_1 is at most

$$\frac{|qr|}{2c} \;\leqslant\; \frac{3\,\text{width}_e(f) \cdot \tan\theta_{\max}}{c}$$

$\square$

Using lemma 3 it is easy to show that $K(e)$, the number of visible intersections of an edge e of $\mathcal{T}$ with edges in front of it, is expected to be small.

Lemma 4

$$\text{E}[K(e)] \leqslant \frac{3\,\text{width}_e(e) \cdot \tan\theta_{\max}}{c} + 2.$$

Using that $\text{width}_e(e) \leqslant \sigma$ and $\tan\theta_{\max} \leqslant \tan(\theta) + \frac{2c}{\sin\phi}$ (by Lemma 1) we obtain our final result:

Theorem 5 *Let $\mathcal{T}$ be a terrain of n triangles with fatness ϕ, steepness θ, and scale factor σ. Then a visibility map of $\mathcal{T}$ under perspective projection has smoothed complexity*

$$O\left(\left(\frac{\tan\theta}{c} + \frac{1}{\sin\phi}\right)\sigma n\right)$$

when adding noise to each vertex's elevation that is uniformly distributed in an interval $[-c, c]$, with c a fixed constant fraction of the minimum edge length of the underlying triangulation.

3 Concluding remarks

We proved that the smoothed complexity of the visibility map of not-too-steep terrains with fat triangles of similar size is $O(n)$. This is the first time that realistic input models are combined with smoothed analysis. Such an approach could also shed light on the complexity of other terrain structures. For example, the complexity of the river network on real-world terrains seems to be linear, while the worst-case complexity of the river network on a terrain with the above-mentioned properties is still $\Theta(n^2)$ [7]. Combining these properties with smoothed analysis may lead to better bounds.

References

[1] B. Aronov, M. de Berg, S. Thite. *The Complexity of Bisectors and Voronoi Diagrams on Realistic Terrains.* In ESA 2008, pp 100-111.

[2] S. Chauduri, V. Koltun. *Smoothed analysis of probabilistic roadmaps.* Computational Geometry: Theory and Applications, to appear.

[3] V. Damerow, F. Meyer auf der Heide, H. Räcke, C. Scheideler, C. Sohler. *Smoothed Motion Complexity.* In ESA 2003, pp 161-171.

[4] V. Damerow, C. Sohler. *Extreme Points Under Random Noise.* In ESA 2004, 264-274.

[5] M. de Berg, A.F. van der Stappen, J. Vleugels,M.J. Katz. *Realistic Input Models for Geometric Algorithms.* Algorithmica, 34(1):81-97,2002.

[6] M.J. Katz, M.H. Overmars, M. Sharir. *Efficient Hidden Surface Removal for Objects with Small Union Size.* Comput. Geom., 2:223-234,1992.

[7] M. de Berg, O. Cheong, H. Haverkort, J.G. Lim, L. Toma. *I/O-efficient flow modeling on fat terrains.* In WADS, pp 239-250, 2007.

[8] G.Mandlburger, C. Briese. *Using Airborne Laser Scanning for Improved Hydraulic Models.* In International Congress on Modelling and Simulation 2007, pp 731-738.

[9] E. Moet, M. van Kreveld, A.F. van der Stappen. *On realistic terrains.* In SoCG 2006, pp 177-186.

[10] D.A. Spielman, Shang-Hua Teng. *Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time.* J. ACM, 51(3):385-463, 2004.

[11] D.A. Spielman. *http://www.cs.yale.edu/homes/spielman/SmoothedAnalysis/index.html*

Planar Visibility Counting[*]

M. Fischer M. Hilbig C. Jähn F. Meyer auf der Heide M. Ziegler,
Heinz Nixdorf Institute and University of Paderborn, GERMANY

Abstract

For a fixed virtual scene (=collection of simplices) $\mathcal{S}$ and given observer position $\vec{p}$, how many elements of $\mathcal{S}$ are weakly visible (i.e. not fully occluded by others) from $\vec{p}$? The present work explores the trade-off between query time and preprocessing space for these quantities in 2D: exactly, in the approximate deterministic, and in the probabilistic sense. We deduce the *existence* of an $\mathcal{O}(m^2/n^2)$ space data structure for $\mathcal{S}$ that, given $\vec{p}$ and time $\mathcal{O}(\log n)$, allows to approximate the ratio of occluded segments up to arbitrary constant absolute error; here m denotes the size of the Visibility Graph—which may be quadratic, but typically is just linear in the size n of the scene $\mathcal{S}$. On the other hand, we present a data structure *constructible* in $\mathcal{O}\big(n\cdot\log(n)+m^2\cdot\operatorname{polylog}(n)/\ell\big)$ preprocessing time and space with similar approximation properties and query time $\mathcal{O}(\ell\cdot\operatorname{polylog} n)$, where $1\leq\ell\leq n$ is an arbitrary parameter.

1 Motivation, Introduction, Overview

In computer graphics, occlusion culling algorithms exhibit both a) the potential of drastically accelerating the rendering of virtual scenes (namely in case when most objects are occluded and therefore need not be drawn) as well as b) an overhead for selecting those objects which are at least partially visible and thus have to be sent to the graphics hardware (conservative approach). Particularly for scenes and observer viewpoints where 'most' is visible, the benefit of a) may be rather low and due to b) result in a net performance *loss*. One would like to estimate in advance whether occlusion culling actually is going to pay off, and if not, turn it off. More generally our goal is to devise algorithms that automatically and optimally adapt, on a per-scene and per-frame basis, to the trade-off between spending more/less computational effort in filtering occluded objects on the one hand and the gain/loss in reduced rendering time on the other hand.

We propose the number of drawing primitives at least partly visible from a given observer position as a quantitative measure for estimating, in comparison with the total number of primitives (i.e. the *visibility ratio*), how much conservative occlusion culling can reduce the rendering complexity.

Definition 1 (visibility count) *For a scene* $\mathcal{S}=\{S_1,\ldots,S_n\}$ *of 'geometric primitives'* $S_i\subseteq\mathbb{R}^d$*, a subset of 'targets'* $\mathcal{T}\subseteq\mathcal{S}$*, and an observer position* $\vec{p}\in\mathbb{R}^d$*, let* $\mathcal{V}(\mathcal{S},\vec{p},\mathcal{T}):=\{T\in\mathcal{T}\,|\,\exists\vec{q}\in T:\forall S\in\mathcal{S}\setminus\{T\}:[\vec{p},\vec{q}]^\circ\cap S=\emptyset\}$ *and denote by* $\mathrm{V}(\mathcal{S},\vec{p},\mathcal{T}):=\operatorname{Card}\mathcal{V}(\mathcal{S},\vec{p},\mathcal{T})$ *the number of objects in* $\mathcal{T}$ *visible (i.e. not fully occluded) from* $\vec{p}$ *through* $\mathcal{S}$*. Here* $[\vec{p},\vec{q}]^\circ:=\{\lambda\vec{p}+(1-\lambda)\vec{q}:0<\lambda<1\}$ *means the relatively open straight line segment from* $\vec{p}$ *to* $\vec{q}$*.*

The present work describes a new randomized algorithm for approximating the visibility count $\mathrm{V}(\mathcal{S},\vec{p},\mathcal{S})$ for 2D scenes of non-intersecting line segments. We presume $\mathcal{S}$ to be arbitrary but fixed, whereas the observer position $\vec{p}$ is given as input. In view of the size $N=\operatorname{Card}(\mathcal{S})$ of the virtual scenes and the frame rates aimed at, running times must be sublinear $\mathcal{O}(N^\alpha)$, $\alpha<1$; after preprocessing $\mathcal{S}$ into data structures of almost linear space $\mathcal{O}(N^{1+\epsilon})$, $\epsilon\ll 1$. Here (time and) space complexity refers to the *number* of (operations on) unit-size real coordinates used (performed) by an algorithm—as opposed to e.g. rationals of varying bitlength. Our algorithm features a parameter $1\leq\ell<n$ to trade preprocessing space for query time.

2 Exact Visibility Counting

Visibility comprises a highly active field of research, both heuristically and in the sound framework of computational geometry [CCSD03]. Particularly the latter has proven combinatorially and algorithmically non-trivial already in the plane [ORou87, Ghos07]. Here the case of (simple) polygons is well studied [CAF07]; and so is point–point, point–segment, and segment–segment visibility for scenes $\mathcal{S}$ of n non-crossing line segments, captured e.g. in the Visibility Graph data structure. Our observer positions $\vec{p}$, however, are *not* restricted to segments in $\mathcal{S}$.

Fact 2 *Given a collection* $\mathcal{S}$ *of* n *non-crossing line segments in the plane and an observer position* $\vec{x}$*,* $\mathrm{V}(\mathcal{S},\vec{x},\mathcal{S})$ *can be calculated in time* $\mathcal{O}(n\cdot\log n)$ *and space* $\mathcal{O}(n)$*.*

[*]Supported by DFG projects Me872/12-1 within SPP 1307, by Zi1009/1-2, and by DFG Research Training Group GK-693 of the *Paderborn Institute for Scientific Computation* (PaSCo). The last author is grateful to 정지원 (né Otfried Schwarzkopf) for the opportunity to visit KAIST. A full version of this extended abstract including proofs and figures is available at arxiv.org/abs/0810.0052.

Also visibility *reporting* algorithms have to be ruled out for cases of linear output size. Our goal is *counting* in *sub*linear time. Even logarithmic time does become easily feasible based on the locus approach of storing *all* visibility counts in a Visibility Space Partition (VSP) [Schi01].

Fact 3 *a) For a collection $\mathcal{S}$ of n non-crossing line segments in the plane, there exists a partition of $\mathbb{R}^2$ into $\mathcal{O}(n^4)$ convex cells such that, for all observer positions $\vec{x} \in C$ within one cell C, $\mathcal{V}(\mathcal{S}, \cdot, \mathcal{S})$ is the same.*

b) The data structure indicated in a) and including for each cell its corresponding visibility count $\mathrm{V}(\mathcal{S}, C, \mathcal{S})$ uses storage $\mathcal{O}(n^4 \cdot \log n)$ and can be computed in time $\mathcal{O}(n^4 \cdot \log^2 n)$. Then, given an observer position $\vec{x}$, its corresponding cell $C \ni \vec{x}$ can be located, and the associated visibility count identified, in time $\mathcal{O}(\log n)$.

Proof. of Claim a): Draw lines through all $\binom{2n}{2}$ pairs of the $2n$ segment endpoints. It is easy to see that, in order for a near segment to appear in sight, the observer has to cross one of these $\mathcal{O}(n^2)$ lines; compare Lemma 7 below. Hence, within each of the $\mathcal{O}(n^4)$ cells they induce, the subset of segments weakly visible remains the same. □

2.1 Size of Visibility Space Partitions

The quartic size bound of Fact 3a) is of course prohibitive—and sharp in the worst case. In order to avoid trivialities, we restrict to *non*degenerate segment configurations $\mathcal{S}$ where i) any two segments meet only in their common endpoints, ii) no three endpoints share a common line, and iii) any two lines, defined by pairs of endpoints, do meet.

We have already referred to (and implicitly employed in Fact 3 a refinement of) the Visibility Space Partition; so here finally comes the formal

Definition 4 *For two non-degenerate collections $\mathcal{S}$ and $\mathcal{T}$ of segments in the plane, partition all viewpoints $\vec{p} \in \mathbb{R}^2$ into classes having equal visibility $\mathcal{V}(\mathcal{S}, \vec{p}, \mathcal{T})$. Moreover let $\mathrm{VSP}(\mathcal{S}, \mathcal{T})$ denote the collection of connected components of these equivalence classes. The size of VSP is the number of line segments forming the boundaries of these components.*

Observe that $\mathrm{VSP}(\mathcal{S}, \mathcal{T})$ indeed constitutes a planar subdivision: a coarsening of the $\mathcal{O}(n^4)$ convex polygons induced by the arrangement of $\mathcal{O}(n^2)$ lines from the proof of Fact 3a). In fact a class of viewpoints of equal visibility can be disconnected and delimited by very many segments, hence merely counting the number of classes or cells does not reflect the combinatorial complexity. Fact 3a) and Proposition 5a) correspond to [Mato02, Exercise 6.1.7].

Proposition 5 *a) Even for a singleton target T, there exist a nondegenerate line segment configurations $\mathcal{S}$ such that $\mathrm{VSP}(\mathcal{S}, \{T\})$ has $\Omega(n^4)$ separate connected components.*

b) To each n, there exists a nondegenerate configuration $\mathcal{S}$ of at least n segments admitting a convex planar subdivision of complexity $\mathcal{O}(n)$ such that, from within each cell, the view to $\mathcal{S}$ is constant; i.e. $\mathrm{VSP}(\mathcal{S}, \mathcal{S})$ has linear size.

c) The size of $\mathrm{VSP}(\mathcal{S}, \mathcal{S})$ is at most quadratic in the size m of the Visibility Graph of $\mathcal{S}$.

d) A data structure as in Fact 3 can be calculated in time $\mathcal{O}(n \cdot \log n + m^2 \cdot \log^2 n)$ and space $\mathcal{O}(m^2)$.

Since the Visibility Graph itself can have at most quadratically more edges than vertices, Item c) strengthens Fact 3a). Empirically we have found that a 'random' scene typically induces a VSP of roughly quadratic size. This agrees with a 'typical' scene to have a linear size Visibility Graph according to [ELPZ07].

2.2 Single Target Visibility: Time versus Space

Intuitively it should be possible to reduce the memory consumption of Fact 3 at the expense of increasing the time bound. We achieve this for the case of one target, that is the decision version of visibility $\vec{x} \mapsto \mathrm{V}(\mathcal{S}, \vec{x}, \{T\}) \in \{0, 1\}$:

Theorem 6 *For each 2D scene $\mathcal{S}$, and line segment T, and $1 \leq \ell \leq n$, an $\mathcal{O}(n^4/\ell)$ size data structure can be computed in time $\mathcal{O}(n^4 \cdot \log^2 n/\ell)$ that allows to decide, given $\vec{p}$, whether T is weakly visible from $\vec{p}$ through $\mathcal{S}$, in query time $\mathcal{O}(\ell \cdot \log n)$.*

Note that Fact 3 is included as $\ell = 1$; whereas for $\ell := n^{1-\epsilon}$ we get arbitrarily close to cubic space while maintaining sublinear query time. On the other end, no value of ℓ recovers Fact 2.

Lemma 7 *Fix a collection $\mathcal{S} \uplus \{T\}$ of $n+1$ non-crossing segments in the plane. Let $L_1, \ldots, L_k$ denote the $k = \binom{2n+2}{2}$ lines induced by the pairs of endpoints of segments in $\mathcal{S} \cup \{T\}$. For an observer moving in the plane, the weak visibility of T can change only as she crosses*

- *either one of the lines L_i intersecting T*
- *or someline supporting a segment $S \in \mathcal{S}$.*

Proof. Standard continuity argument: Let $\vec{p}$ denote the observer's position and suppose point $\vec{x} \in T$ is visible, i.e. the segment $[\vec{p}, \vec{x}]$ does not intersect $S \in \mathcal{S}$. Now move $\vec{p}$ until $\vec{x}$ is just about to become hidden behind $S \in \mathcal{S}$. Then start moving $\vec{x}$ on T such as to remain visible. Keep moving $\vec{p}$ and adjusting $\vec{x}$: this is possible (at least) as long as the line through $\vec{p}$ and $\vec{x}$ avoids all endpoints of $\mathcal{S} \cup \{T\}$. □

Proof. [Theorem 6] Consider, as in the proof of Fact 3, the $\mathcal{O}(n^2)$ lines induced by pairs of segment endpoints of $\mathcal{S}$. Consider the intersections of these lines with T (if any). Partition T into $\mathcal{O}(\ell)$ subsegments $T_1, \ldots, T_\ell$, each intersecting $\mathcal{O}(n^2/\ell)$ of the above lines. For each piece T_i, take the arrangement $\mathcal{A}_i$ of size $\mathcal{O}\big((n^2/\ell + n)^2\big)$ induced by those lines intersecting T_i, and all $\mathcal{O}(n)$ lines through one endpoint of T_i and one of some $S \in \mathcal{S}$, and all $\mathcal{O}(n)$ lines supporting segments from $\mathcal{S}$. By Lemma 7, within each cell C of $\mathcal{A}_i$, the weak visibility of T_i is constant (either `yes` or `no`) and can be stored with C: Doing so for each $\mathcal{A}_i$ $(1 \leq i \leq \ell \leq n)$ and each of the $\mathcal{O}(n^4/\ell^2 + n^3/\ell + n^2)$ cells C of $\mathcal{A}_i$ uses memory of order $\mathcal{O}(n^4/\ell + n^3 + n^2\ell) = \mathcal{O}(n^4/\ell)$ as claimed; and corresponding time according to Fact 3b).

Then, given a query point $\vec{p} \in \mathbb{R}^2$, locating $\vec{p}$ in each arrangement $\mathcal{A}_i$ takes total time $\mathcal{O}(\ell \cdot \log n)$; and yields the answer to whether T_i is weakly visible from $\vec{p}$ or not. Now T itself is of course visible iff some T_i is: a disjunction computable in another $\mathcal{O}(\ell)$ steps. □

Similar to Proposition 5d), we can improve Theorem 6 to $\mathcal{O}(n \cdot \log n + m^2 \cdot \log^2 n/\ell)$ preprocessing time and $\mathcal{O}(m^2/\ell)$ space and query time $\mathcal{O}(\ell \cdot \log n)$, where m denotes the size (number of edges) of the Visibility Graph of $\mathcal{S}$.

3 Approximate Visibility Counting

Lacking deterministic exact algorithms for calculating visibility counts satisfying both time and space requirements, we now resort to approximations: of $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{S})$ up to prescribable absolute error $k \in \mathbb{N}$ or, equivalently, of the visibility *ratio* $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{S})/\operatorname{Card}(\mathcal{S})$ up to absolute error $\epsilon = k/\operatorname{Card}(\mathcal{S})$. Our main result presents a randomized approximation within sublinear query time using almost cubic preprocessing space and time in the worst-case; almost linear space and quadratic preprocessing time in the 'typical' case.

3.1 Deterministic Approach: Relaxed VSPs

Visibility space partitions, and the algorithms based upon them, are so memory expensive because they distinguish (i.e. introduces separate arrangement cells for) observer positions whose visibility differs by as little as one; recall Definition 4. Considerably more space efficient algorithms are feasible by partitioning observer space into (or merely covering it by) more coarse classes:

Definition 8 *Fix $k \in \mathbb{N}$ and collections $\mathcal{S}$ and $\mathcal{T}$ of non-intersecting segments in the plane. Some covering $\{C_1, \ldots, C_I\}$ of $\mathbb{R}^2$ is a k-relaxed VSP of $(\mathcal{S}, \mathcal{T})$ if*

$$\forall 1 \leq i \leq I \; \forall \vec{p}, \vec{q} \in C_i : \quad \mathrm{V}(\mathcal{S}, \vec{p}, \mathcal{T}) - \mathrm{V}(\mathcal{S}, \vec{q}, \mathcal{T}) \;\leq\; k.$$

In the sequel we shall restrict to k-relaxed VSPs which constitute planar subdivisions (i.e. each C_i being a simple polygon); and refer to their size in the sense of Definition 4. Indeed, such VSPs allow for locating a given observer position $\vec{x}$ in logarithmic time to yield a cell $C_i \ni \vec{x}$ which, during preprocessing, had been assigned a value $\mathrm{V}(\mathcal{S}, \vec{q}, \mathcal{T})$ approximating $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{T})$ up to absolute error at most k.

Observe that, for $\operatorname{Card}\mathcal{T} \leq k$, the trivial planar subdivision $\{\mathbb{R}^2\}$ is a k-relaxed VSP of $(\mathcal{S}, \mathcal{T})$. In particular the quartic lower size bound of Proposition 5b) applies only to 0-relaxed VSPs but breaks down for $k \geq 1$. This indicates that much smaller sizes become feasible when considering k-relaxed VSPs for, say, $k \approx \sqrt{n}$ or even $k \approx n/\log n$.

Proposition 9 a) *To $k < n-1$ there exists a non-degenerate family $\mathcal{S}$ of n segments in the plane such that any k-relaxed VSP has size $\Omega(n^2/k)$.*

b) *There also exist such families such that any k-relaxed VSP has size at least $\Omega(n^4/k^4)$.*

c) *Let $\mathcal{S}$ be a non-degenerate family of n segments in the plane and N the size of its VSP. Then there exists a k-relaxed VSP of size $\lfloor N/(k+1) \rfloor$.*

d) *There also exists a k-relaxed VSP of size $\mathcal{O}(m^2/k^2)$, where m denotes the size of the Visibility Graph of $\mathcal{S}$.*

Recall that $N \leq m^2 \leq n^4$, thus leaving quadratic gap between a) and d) for k small; and between b) and d) for k large. Item c) succeeds over d) in cases where N asymptotically does not exceed m^2/k.

Proposition 9d) is an application of the Cutting Lemma of Chazelle, Friedman, and Matoušek [Mato02, Lemma 4.5.3]. Notice that we only claim the *existence* of such small data structures. In order to *construct* them, our proofs of Proposition 9c+d) both proceed by first calculating the 0-relaxed VSP and then coarsening it. Specifically, although a Triangular Cutting can be found in asymptotically nearly optimal time [Agar90], 'filling' it with visibility counts seems to incur quartic preprocessing time, again—not to mention the impractically large constants hidden in asymptotic big-Oh notation.

3.2 Random Sampling

We now proceed to the simple, generic, randomized

Algorithm 10

i) *Guess a sample target $\mathcal{T} \subseteq \mathcal{S}$ of size m.*

ii) *Calculate the count $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{T})$ of objects in $\mathcal{T}$ visible through $\mathcal{S}$.*

iii) *Return the ratio $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{T})/\operatorname{Card}(\mathcal{T})$;*

iv) *and hope that it does not deviate too much from the 'true' ratio $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{S})/\operatorname{Card}(\mathcal{S})$.*

Item iv) is justified for instance by the following application of the Chernoff–Hoeffding Bound [AlSp00]:

Lemma 11 *Fix $\vec{x} \in \mathbb{R}^d$ and $\delta > 0$, then choose $\mathcal{T} \subseteq \mathcal{S}$ as m independent identically distributed random draws from $\mathcal{S}$. It holds*

$$\textbf{\textit{Prob}}_{\mathcal{T}}\Big[\big|\tfrac{\mathrm{V}(\mathcal{S},\vec{x},\mathcal{T})}{m} - \tfrac{\mathrm{V}(\mathcal{S},\vec{x},\mathcal{S})}{n}\big| \geq \delta\Big] \leq 2 \cdot e^{-2m\cdot\delta^2}$$

In other words: In Algorithm 10 taking m (quadratic in the aimed *absolute* accuracy δ but) *constant* with respect to the scene size n suffices to achieve the desired approximation with constant probability; slightly increasing it further amplifies exponentially the chance for success.

3.3 The VC-Dimension of Visibility

Note that the random experiment $\mathcal{T}$ and the probability analysis of its properties in Lemma 11 holds for each $\vec{x}$ but not uniformly in $\vec{x}$. This means for our purpose to re-sample $\mathcal{T} \subseteq \mathcal{S}$ at every frame. On the other hand, Lemma 11 does not exploit any geometry. An important connection between combinatorial sampling and geometric properties is captured by the Vapnik–Chervonenkis Dimension [AlSp00]:

Fact 12 *Let X be a set and $\mathcal{R}$ a collection of subsets $R \subseteq X$. Denote* $d := \mathrm{VCdim}(X, \mathcal{R}) :=$

$$\max\big\{\operatorname{Card} Y \big| Y \subseteq X, \{Y \cap R : R \in \mathcal{R}\} = 2^Y\big\}. \quad (1)$$

Let $Y \subseteq X$ be randomly distributed at uniform of $\operatorname{Card}(Y) \geq \Omega\big((d \cdot \log\frac{d}{\delta} + \log\frac{1}{p})/\delta^2\big)$. *Then with probability at least $1 - p$ it holds for each $R \in \mathcal{R}$:* $\big|\operatorname{Card}(X \cap R)/\operatorname{Card}(X) - \operatorname{Card}(Y \cap R)/\operatorname{Card}(Y)\big| \leq \delta$.

Lemma 13 *Fix a collection $\mathcal{S}$ of n noncrossing $(d-1)$–dimensional simplices in $\mathbb{R}^d$.*

a) Define $X := \mathcal{S}$ and $\mathcal{R} := \{\mathcal{V}(\mathcal{S}, \vec{x}, \mathcal{S}) : \vec{x} \in \mathbb{R}^d\}$. Then $\mathrm{VCdim}(X, \mathcal{R}) \leq d^2 \cdot \big(\log n + \mathcal{O}(1)\big)$.

b) In $\mathbb{R}^2$ there exist non-degenerate collections $\mathcal{S} = X$ of n line segments such that $\mathrm{VCdim}(X, \mathcal{R}) \geq \log n - \mathcal{O}(\operatorname{loglog} n)$.

Proof. [Claim a] Fact 3a) straight forwardly generalizes to yield $\operatorname{Card}\mathcal{R} \leq \mathcal{O}(n)^{d^2}$. Hence, in Equation (1), $2^Y = \{Y \cap R : R \in \mathcal{R}\}$ requires $2^{\operatorname{Card} Y} \leq \operatorname{Card}\mathcal{R}$ and therefore $\operatorname{Card} Y \leq d^2 \cdot \big(\log n + \mathcal{O}(1)\big)$. □

3.4 Main Result

Lemma 13a), together with Fact 12, enhances Lemma 11: For $d := 2$, a sample $\mathcal{T}$ of size $m := \mathcal{O}(\operatorname{polylog} n)$ has visibility ratio $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{T})/m$ close to the 'true' on $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{S})/n$ with high probability with respect to *all* viewpoints $\vec{x}$! In particular we may preprocess the visibility of each $T \in \mathcal{T}$ separately according to Theorem 6 and conclude in combination with Proposition 5d):

Theorem 14 *Given $0 < \delta < 1$, a collection $\mathcal{S}$ of n non-crossing segments in the plane ($d = 2$), and $1 \leq \ell \leq n \leq m$ where m denotes the size of the Visibility Graph of $\mathcal{S}$. Then a randomized algorithm can preprocess $\mathcal{S}$ within time $\mathcal{O}\big(n \cdot \log n + m^2 \cdot \operatorname{polylog} n \cdot \log\frac{1}{\delta}/(\ell\cdot\delta^2)\big)$ and space $\mathcal{O}\big(m^2 \cdot \operatorname{polylog} n \cdot \log\frac{1}{\delta}/(\ell\cdot\delta^2)\big)$ into a data structure having with high probability the following property: Given $\vec{x} \in \mathbb{R}^2$, one can approximate the visibility ratio $\mathrm{V}(\mathcal{S}, \vec{x}, \mathcal{S})/\operatorname{Card}(\mathcal{S})$ up to absolute error at most δ in time $\mathcal{O}\big(\ell \cdot \operatorname{polylog} n \cdot \log\frac{1}{\delta}/\delta^2\big)$.*

Recall the trade-off between space and time gauged by parameter ℓ; and m being 'typically' linear in n.

4 Conclusion and Perspective: Dimension > 2

For constant δ, $k := \delta \cdot n$ in Proposition 9d) yields logarithmic query time in worst-case quadratic ('typical' case even linear) preprocessing space: this beats Theorem 14. On the other hand, the simplicity of Algorithm 10 makes it very practical; and preprocessing according to Theorem 6 is reasonable as well. In fact we have implemented the algorithm underlying Theorem 14 and are currently evaluating it for the purpose of adaptive culling as mentioned in the introduction.

The present work restricts to the planar case of line segments. Many virtual scenes are $2\frac{1}{2}$-dimensional: objects of various heights rooted on a common plane. In 3D scenes of triangles, however, visibility immediately becomes 3SUM-complete and thus unlikely tractable in time $o(n^2)$, see [GaOv95, Section 6.1].

References

[Agar90] P. Agarwal: "Partitioning Arrangements of Lines I: An Efficient Deterministic Algorithm", pp.449–483 in *Discrete Comput. Geom.* vol.**5** (1990).

[AlSp00] N. Alon, J.H. Spencer: "*The Probabilistic Method*", 2nd Edition, Wiley (2000).

[CAF07] S. Charneau, L. Aveneau, L. Fuchs: "Exact, Robust and Efficient Full Visibility Computation in Plücker Space", pp.773–782 in *Visual Comput.* vol.**23** (2007).

[CCSD03] D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, F. Durand: "A Survey of Visibility for Walkthrough Applications", pp.412–431 in *IEEE Transactions on Visualization and Computer Graphics* vol.**9:3** (2003).

[ELPZ07] H. Everett, S. Lazard, S. Petitjean, L. Zhang: "On the Expected Size of the 2D Visibility Complex", pp.361–381 in *Int. J. Comput. Geom.* vol.**17:4** (2007).

[GaOv95] A. Gajentaan, M. Overmars: "On a Class of $\mathcal{O}(n^2)$ Problems in Computational Geometry", pp.165–185 in *Computat. Geom.: Theory & Applications* vol.**5:3** (1995).

[Ghos07] S.K. Ghosh: "*Visibility Algorithms in the Plane*", Cambridge University Press (2007).

[Mato02] J. Matoušek: "*Lectures on Discrete Geometry*", Springer Graduate Texts in Mathematics vol.**212** (2002).

[ORou87] J. O'Rourke: "*Art Gallery Theorems and Algorithms*", Oxford University Press (1987).

[Schi01] R.D. Schiffenbauer: "A Survey of Aspect Graphs", *TR-CIS-2001-01* Brooklyn University (2001).

On the Limitations of Combinatorial Visibilities

Y. Disser[*] D. Bilò[*] M. Mihalák[*] S. Suri[†] E. Vicari[*] P. Widmayer[*]

Abstract

We consider *combinatorial visibilities* of vertices of a polygon $\mathcal{P}$, a *local* view of $\mathcal{P}$ from its vertices, and ask how much *global* combinatorial information they provide about $\mathcal{P}$. We study three related questions about the connection of visibility graphs and combinatorial visibilities, and show that in no case the combinatorial visibilities carry enough information.

1 Introduction, Model, and Contribution

In microrobotics, there is a strong tendency to deploy lots of cheap and simple microrobots (instead of few complex, expensive ones) to perform robotic tasks such as the exploration of an unknown environment. This gives rise to studies about the sensory and motor capabilities that microrobots need for carrying out such tasks [4, 1]. One particular model [3] assumes that robots (modeled as points) in a polygonal environment have no notion of (or no way of measuring) coordinates or distances. Instead, these robots are limited to seeing vertices of the polygon that are visible from their location when polygon edges are treated as opaque walls, and seeing whether visible vertices are connected with a polygon edge. This concept is usually referred to as "combinatorial visibility". In a convex polygon, for instance, a robot on any of the vertices "sees" all other vertices and edges (and this is not true in any other polygon). We add that in this model the robots can only move from a vertex to any visible vertex. Since this combinatorial visibility appears to be quite powerful [3], there was hope that the combinatorial visibilities of all vertices together might be enough to derive all the combinatorial (i.e., non-geometric) information about the polygon – the "map". The map (also called "visibility graph" [2]) of the polygon is defined as follows: The map (visibility graph) consists of a vertex for each polygon vertex, and has an edge between any two mutually visible vertices. For a convex polygon, for instance, the visibility graph is the complete graph and can easily be inferred from all combinatorial visibilities together.

In this paper, we show that all combinatorial visibilities of a simple polygon are in fact not enough to infer the map (Theorem 1). For certain robotic tasks, such as the meeting of two robots in an unknown polygon, a map might not be needed, for instance if the simple polygon looks nonperiodic to the robot(s). For periodic-looking polygons, there was hope that the vertices seen from a given vertex and those seen from a "periodic partner" of the given vertex (details follow) would themselves be periodic partners; this property would have allowed a variety of tasks to be solved. We show that this is, unfortunately, not the case (Theorem 2). For map computation, the question arises what to add to the abilities of microrobots to make it possible. We show that adding the "local" ability for a robot to measure the polygon angle at its current vertex is not enough (Theorem 3). The common theme behind these results is the relation between combinatorial visibilities and the well-studied visibility graph concept.

In this work we only consider simple polygons. We denote the n vertices of a simple polygon $\mathcal{P}$ by $V=\{v_0, v_1, \cdots, v_{n-1}\}$, ordered along the boundary in counterclockwise (ccw) order. The polygon has n edges $E = \{e_0, e_1, \ldots, e_{n-1}\}$, where $e_i = (v_i, v_{i+1})$, $i = 0, \ldots, n-1$.[1] The *combinatorial visibility* of a vertex v is given by a binary vector who's j-th component encodes whether the j-th visible vertex and the $(j+1)$-th visible vertex form an edge of $\mathcal{P}$ or not, we call this a *combinatorial visibility vector* $\mathrm{cvv}(v)$. The following definitions capture this more formally. Consult Fig. 1 along with the definitions.

Definition 1 *Two vertices $v_i, v_j \in V$ form a* visible pair *in $\mathcal{P}$, if the line segment $\overline{v_i v_j}$ lies entirely within $\mathcal{P}$ (in particular, v_i forms a visible pair with itself for any i). We say v_i and v_j* see each other *and write $v_i \leftrightarrow_{\mathcal{P}} v_j$. We drop the index $\mathcal{P}$ and simply write '$\leftrightarrow$', if the corresponding polygon $\mathcal{P}$ is clear from the context.*

Definition 2 *We define* $\mathrm{view}(v_i)$ *of vertex v_i in $\mathcal{P}$, the view of vertex v_i, to be the set of vertices that v_i sees in $\mathcal{P}$. Formally,*

$$\mathrm{view}(v_i) := \{\, v_j \in V \,|\, v_i \leftrightarrow v_j \,\} .$$

We write $\mathrm{view}_j(v_i)$ to denote the j-th vertex, $j \geq 0$, that v_i sees in ccw order, starting at v_i itself, both $\mathrm{view}_0(v_i)$ and $\mathrm{view}_{|\mathrm{view}(v_i)|}(v_i)$ denoting v_i.

[*]Institute of Theoretical Computer Science, ETH Zurich, {ydisser|dbilo|mmihalak|vicariel|widmayer}@inf.ethz.ch

[†]Department of Computer Science, University of California, Santa Barbara, suri@cs.ucsb.edu

[1]We consider all operations on indices modulo n.

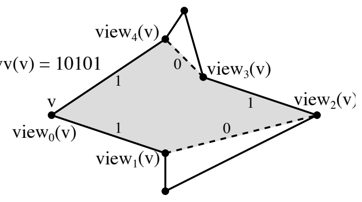


Figure 1: Illustration of a combinatorial visibility vector.

Definition 3 *The* combinatorial visibility vector $\mathrm{cvv}(v_i) \in \{0,1\}^{|\mathrm{view}(v_i)|}$ *of vertex $v_i \in V$ is a binary vector with the j-th element, $j \geq 0$, given by*

$$\mathrm{cvv}_j(v_i) = \begin{cases} 1, & \text{if } (\mathrm{view}_j(v_i), \mathrm{view}_{j+1}(v_i)) \in E, \\ 0, & \text{else.} \end{cases}$$

Note that $\mathrm{view}_1(v_i) = v_{i+1}$ and $\mathrm{view}_{|\mathrm{view}(v_i)|-1}(v_i) = v_{i-1}$ as every vertex sees its neighboring vertices on the polygon boundary. Therefore $\mathrm{cvv}_0(v_i) = \mathrm{cvv}_{|\mathrm{view}(v_i)|-1}(v_i) = 1$ for all $v_i \in V$.

Definition 4 *The* combinatorial visibility sequence cvs *of $\mathcal{P}$ lists all combinatorial visibility vectors of the individual vertices of $\mathcal{P}$ in ccw order:*

$$\mathrm{cvs} := (\mathrm{cvv}(v_0), \ldots, \mathrm{cvv}(v_{n-1}))\,.$$

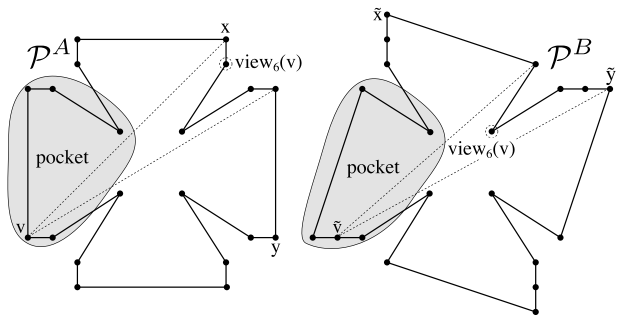
In the next section we first show that the cvs of a polygon $\mathcal{P}$ is not enough to determine the map of $\mathcal{P}$. Based on this, we then show a similar result about visibility in "periodical" polygons. Finally, we show that additionally knowing the angles at the vertices of $\mathcal{P}$ does not determine the visibility graph either.

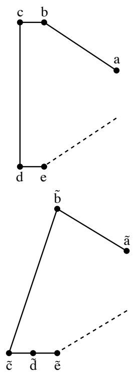
2 Combinatorial Visibility and Visibility Graphs

Theorem 1 *The cvs of a polygon $\mathcal{P}$ does not uniquely define its map.*

Proof. In Fig. 2 we present two polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ that share the same cvs, yet they have different maps. The proof is by an inspection of the polygons assisted by a list of the cvv's and view sequences of the needed vertices.

The idea behind the construction of the polygons is to use multiple copies of a "pocket" of vertices (cf. Fig. 2 for an illustration). Each pocket forms a convex curve, but the vertices connecting the pockets form reflex angles, resulting in a non-convex polygon $\mathcal{P}$. The vertices inside a pocket thus do not see all vertices of $\mathcal{P}$, they see (apart from their own pocket) only parts of exactly two pockets. We use the fact that the vertices have no way to distinguish what pockets they are "looking into" and we modify the polygon $\mathcal{P}^A$ by shifting the vertex c (cf. Fig. 2) so that in $\mathcal{P}^B$ the shifted vertex $\tilde{c}$ looks into different pockets, while not changing the cvv of any vertex. □





vert	cvv	vision sequence
a $\tilde{a}$	1111101111011111	$abcdeadeabcabcde$ $\tilde{a}\tilde{b}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{c}\tilde{d}\tilde{e}$
b $\tilde{b}$	11110111101	$bcdeadeabca$ $\tilde{b}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}$
c $\tilde{c}$	11101111011	$cdeadeabcab$ $\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}$
d $\tilde{d}$	11011110111	$deadeabcabc$ $\tilde{d}\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{c}$
e $\tilde{e}$	10111101111	$eadeabcabcd$ $\tilde{e}\tilde{a}\tilde{c}\tilde{d}\tilde{e}\tilde{a}\tilde{b}\tilde{a}\tilde{b}\tilde{c}\tilde{d}$

Figure 2: Top: Two polygons P^A and P^B with $n = 20$, identical cvs and different visibility graphs. Bottom: cvv and vision sequence of every vertex within a pocket.

Definition 5 *We say that a cvs C is* periodical *with a period $p \geq 2$, if $C_i = C_{i+k\cdot\frac{n}{p}}$ for all $0 \leq i < n$ and for all $1 \leq k < p$. For each $0 \leq i < n$ we say $\left\{ v_{i+k\cdot\frac{n}{p}} \,\middle|\, 0 \leq k < p \right\}$ are* periodical partners.

Theorem 1 shows that the cvs is not enough to determine the visibility graph of a polygon. In the following we are interested in a similar question:[2] Assume vertex v_i sees a vertex v_{i+j} as its k-th visible vertex (in ccw order), does vertex $v_{(i+n/2)}$ see vertex $v_{(i+n/2)+j}$ as its k-th visible vertex? The following theorem implies that this is not the case.

Theorem 2 *There is a polygon $\mathcal{P}$ with a periodical* cvs *of period $p \geq 2$ for which we have*

$$\exists v_i \in V\, \exists j\, :\, \mathrm{cvv}(\mathrm{view}_j(v_i)) \neq \mathrm{cvv}\Big(\mathrm{view}_j\Big(v_{i+\frac{n}{p}}\Big)\Big)\,.$$

Proof. We construct a polygon $\mathcal{P}$ with the aforementioned property from the two polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ in Fig. 2. The polygon will have period $p = 2$. At the end of the proof we show how to generalize the construction to arbitrary periods.

The idea of the construction is to "glue" $\mathcal{P}^A$ and $\mathcal{P}^B$ together at vertices v and $\tilde{v}$ of $\mathcal{P}^A$ and $\mathcal{P}^B$, respectively, where v and $\tilde{v}$ are as depicted in Fig. 2. We

[2]A positive answer to this question would have an impact on various interesting problems in the field of simple robots; for example, on the weak version of the *rendezvous* problem in symmetric polygons.

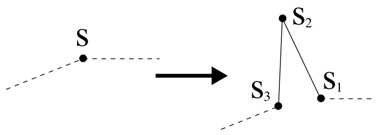


Figure 3: The concept of inserting spikes at vertices.

want to glue the polygons such that every two corresponding vertices w and $\tilde{w}$ of the two polygons form periodical partners in $\mathcal{P}$. Thus, we need to glue the polygons such that the cvv's of corresponding vertices w and $\tilde{w}$ are the same. We can then use the result of Theorem 1 which guarantees the existence of vertices w and $\tilde{w}$ with the same cvv but different views. Formally, if w from $\mathcal{P}^A$ is a vertex v_i in $\mathcal{P}$ and $\tilde{w}$ from $\mathcal{P}^B$ is a vertex $v_{i+n/2}$ in $\mathcal{P}$ (where n is the number of vertices of $\mathcal{P}$), there is a position j in their visions such that $\text{view}_j(v_i) = v_k$ and $\text{view}_j\big(v_{i+\frac{n}{2}}\big) = v_l \neq v_{k+\frac{n}{2}}$. Because of the structure of the two polygons, we will see that $\text{cvv}(v_k) \neq \text{cvv}(v_l)$ which proves the theorem.

We glue both polygons together at a vertex $v \in V_{\mathcal{P}^A}$ and the corresponding vertex $\tilde{v} \in V_{\mathcal{P}^B}$ with the same cvv by splitting these vertices into v, v' and $\tilde{v}, \tilde{v}'$ respectively and merging v with $\tilde{v}'$ and v' with $\tilde{v}$.[3] Because we perform the splitting on both v and $\tilde{v}$, the altered cvv's of the new vertices will still be pairwise equal. Similarly we do not change the cvv of vertices which did not see the split vertices. The problem however is the change in the cvv's of all vertices that see v or $\tilde{v}$ in $\mathcal{P}^A$ or $\mathcal{P}^B$ respectively. Such a vertex now, instead of seeing the original vertex, sees two unconnected vertices (zero in the cvv). Thus, vertices would be able to distinguish between split and non-split vertices in their fields of vision and therefore they would be able to distinguish into which pocket they are "looking".

In order to maintain equal cvs' in both polygons we split all vertices in a similar way. Fig. 3 shows how to split vertices by inserting *spikes*. A spike substitutes a vertex s by three new vertices s_1, s_2, and s_3 as illustrated. It is important that the new vertex s_2 at the tip of the spike does not see any other vertices except for s_1 and s_3. That way, other vertices will have the impression to see two vertices s_1 and s_3 that are not connected by an edge. This is exactly how the spikes resulting from splitting v and $\tilde{v}$ look from outside, so that they are indistinguishable from other spikes. Fig. 4 shows $\mathcal{P}^A$ and $\mathcal{P}^B$ after inserting spikes.[4] Fig. 5 gives a listing of the cvv of each vertex within a pocket. For the sake of brevity we do not give the visibility graph explicitly.

[3] We place the newly created vertices very close together so that if vertices v and x saw each other originally, then after the splitting x sees both v and v'.

[4] Note that we do not assume general position for our construction. However it is easy to modify the spiked versions of our polygons accordingly.

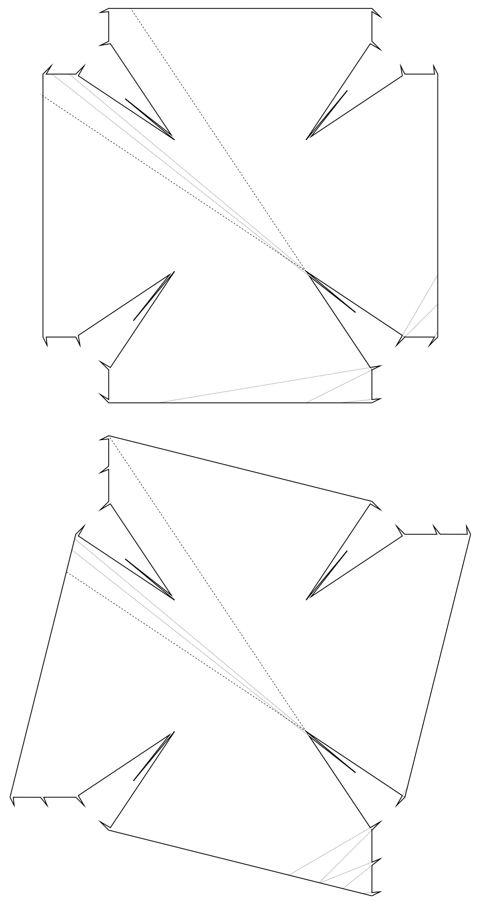
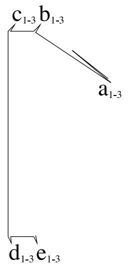


Figure 4: The two polygons from Fig. 2 equipped with spikes and still with identical cvs. The areas visible from different tips are indicated.



vert	cvv
a_1	11001010101001010101 01
a_2	101
a_3	1010101010001010101001010101010111
b_1	11101010100010101010 01
b_2	101
b_3	1010101000101010100111
c_1	1110101000101010100101
c_2	101
c_3	1010100010101010010111
d_1	1110100010101010010101
d_2	101
d_3	1010001010101001010111
e_1	1110001010101001010101
e_2	101
e_3	1000101010100101010111

Figure 5: The combinatorial visibilities of each vertex in a pocket of $\mathcal{P}^A$ after adding spikes (the same cvv's arise for $\mathcal{P}^B$). We write v_{1-3} to denote the group of vertices v_1, v_2, v_3.

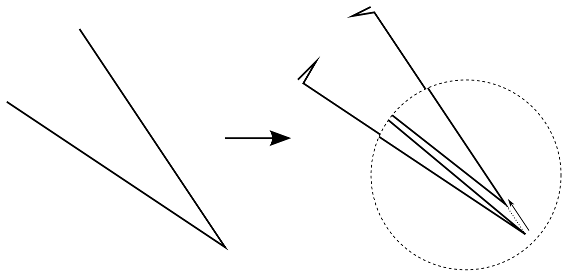


Figure 6: Illustration of how the spikes are inserted at reflex vertices. We chose our modification such that the right neighbor of the spike tip retains the visibility of the original vertex.

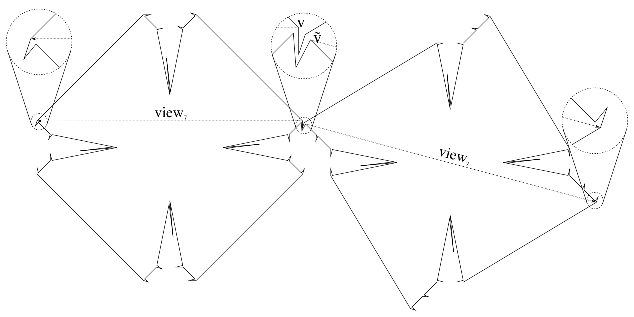


Figure 7: A polygon with $n = 120$ that proves Theorem 2.

For each convex vertex s we are able to insert a spike such that the two new vertices s_1 and s_3 that lie at each side of s_2 have the same vision as s had (apart from seeing s_2 and seeing "gaps" instead of single vertices everywhere else). We do this by splitting the convex vertex along its tangential direction. It is always possible to position the spike tip such that it is seen by its two neighbors only. Because we are able to insert the spike without qualitatively changing the vision of s_1 and s_3, it is obvious that the cvv's of all convex vertices remain equal in $\mathcal{P}^A$ and $\mathcal{P}^B$. For reflex vertices r, however, we cannot introduce a spike such that r_1 and r_3 share the same vision apart from seeing r_2 on different sides. In general we need to split reflex vertices "manually" and make sure that the resulting cvs' of both polygons are equal. Fig. 6 shows how to do this with the four reflex vertices in our case.

Once we have spiked versions of $\mathcal{P}^A$ and $\mathcal{P}^B$, we can glue them together in a straightforward way by simply splitting the spike tip of v and $\tilde{v}$ and attaching the open ends. This modification only affects the cvv of $v_1, \tilde{v}_3$ and $v_3, \tilde{v}_1$ in a similar manner as well as the cvv of the duplicated spike tips $v_2, \tilde{v}_2$ each of which are periodical partners in $\mathcal{P}$. The resulting cvs is still periodical with period 2. Fig. 7 shows the resulting polygon $\mathcal{P}$.

The extension to $p > 2$ is easily made, as we can attach more than two copies of the two spiked polygons

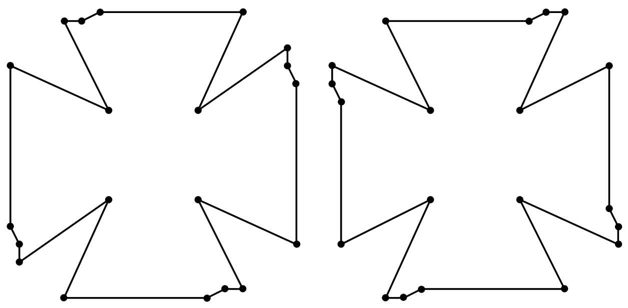


Figure 8: Two polygons with identical cvs and identical interior angles but different G_V. The visibilities are similar to those of $\mathcal{P}^A$ and $\mathcal{P}^B$.

around a common center. □

Theorem 1 shows that the cvs is not sufficient to reconstruct the visibility graph of a polygon. A natural question is to determine a "minimal" structural information we need to add to the cvs for reconstructing the visibility graph. In the following we show that knowing the precise inner angle of every vertex of a polygon in addition to its cvs is still not enough for reconstructing G_V. We show the following theorem.

Theorem 3 *The* cvs *and precise inner angles of a polygon $\mathcal{P}$ do not uniquely determine the map of $\mathcal{P}$.*

Proof. Fig. 8 shows a modified version of the polygons $\mathcal{P}^A$ and $\mathcal{P}^B$ of Fig. 2. As one can easily check, these polygons still have the same cvs and different visibility graphs. In addition, they also have the same set of inner angles at the vertices. The existence of such polygons proves the theorem. □

References

[1] A. Ganguli, J. Cortés, and F. Bullo. Distributed deployment of asynchronous guards in art galleries. In *Proceedings of the American Control Conference*, pages 1416–1421, June 2006.

[2] S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, 1st edition, April 2007.

[3] S. Suri, E. Vicari, and P. Widmayer. Simple robots with minimal sensing: From local visibility to global geometry. *International Journal of Robotics Research*, 27(9):1055–1067, September 2008.

[4] A. Yershova, B. Tovar, R. Ghrist, and S. LaValle. Bitbots: Simple robots solving complex tasks. In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 1336–1341, 2005.

Dynamic Segment Visibility

Mojtaba Nouri Bygi* Mohammad Ghodsi†

Abstract

In this paper we define topological segment visibility, and show how to compute and maintain it as the observer moves in the plane. There are n line segment objects in the plane, and we have a segment observer among them. We first consider static case of the problem, in which the observer and objects are motionless, and then we study dynamic case of the problem, in which the observer can move among obstacles.

1 Introduction

Visibility is an important topic in computer graphics, motion planning, and computational geometry. To deal with the increasing complexity of the scenes considered, some research has been performed in visibility processing in order to accelerate the visibility determination.

Visibility of a point observer has been studied widely in Computer Geometry literature. In visibility computations, the object which is seen along rays (maximal free line segments) is determined. As re-computing visibility of moving object that can be seen along a ray is a time consuming task, some solutions are proposed for this problem. One proposed solution is to classify rays according to their visibility, that is, after finding the object along a given ray, we only need to identify the set of rays that contains the given ray and read the visibility properties of the specified rays.

In this paper we define combinatorial (or topological) segment visibility, and show how to compute and maintain it as the segment moves in the plane. Topology deals with the classification of spaces that are the same up to some equivalence relation [5].

We assume that there are n non-intersection line segment objects in the plane, and we have a line segment observer among them. We say that the observer *sees* an object, if and only if there is a point in the observer that can see the object.

*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, nouribaygi@ce.sharif.edu

†Department of Computer Engineering, Sharif University of Technology, and IPM School of Computer Science (No. CS1382-2-02), Tehran, Iran, ghodsi@sharif.edu

2 Segment Visibility

In this section we will define combinatorial (or topological) segment visibility, and show how to compute and maintain it as the segment moves in the plane. Assume that there are n non-intersection line segment objects in the plane, and a line segment observer among them. We say that the observer *sees* an object, if and only if there is a point in the observer that can see the object. This definition differs from other possible segment-visibility definitions in that here we just want to know whether an object is visible from the segment observer, but the exact portion of the object seen from the observer is not important. This kind of visibility definition is mainly practical in global visibility studies. As the topological visibility of a line segment has not been studied before, we first consider static case of the problem, in which the observer and objects are static. Then we study dynamic case of the problem, in which the observer can move among obstacles.

2.1 Static Case

First we consider the static case, in which the observer AB doesn't move in the scene along the objects (see Figure 1). First we build the visibility graph for the given n line segments. The visibility edges, or discontinuity lines, divide the plane to some regions with constant visibility. Consider a view point p with initial location at the segment endpoint A. Here we use the method of [4] that uses visibility complex of the scene to compute and maintain the visibility from a viewpoint. Depending on the method used to find the crossed edges, the view can be computed in $O(v \log n)$ time or $O(n \log n)$ time (v is the size of the view). We move along the segment toward the other endpoint B. As long as we are on the same region, the visibility will not change. When we cross a discontinuity line, the visibility will change and we need to update the view. So, after the computation of v_0, the initial view around a point p, and a preprocessing step done in $O(v_0 \log v_0)$ time, the view along a trajectory can be maintained in $O(\log v)$ time at each change of visibility [4]. So, if the number of discontinuity lines that intersect the visibility segment is k, the total time needed to compute the view of visibility segment will be $O((n + k) \log n)$. Considering the time needed to construct the visibility complex of the scene, we have the following proposition.

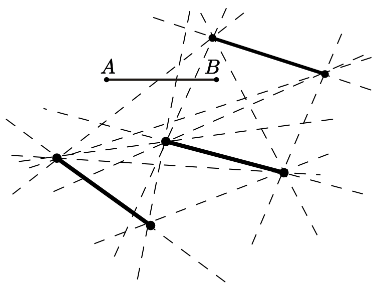


Figure 1: A line segment observer among line segment objects.

Proposition 1 *After the computation of the visibility complex of a scene that composed of n line segments, and a preprocessing step done in $O(m+n\log n)$ time, where m is the number of discontinuity lines, the view from a line segment can be computed in $O((n+k)\log n)$, where k is the number of discontinuity lines that intersect the line segment.*

2.2 Dynamic Case

Now we consider the dynamic case of the problem: A line segment observer s is moving among a set S of n line segment objects. We want to maintain the visibility of the observer as it moves. Our approach is to compute the initial visibility of the observer, as done in Section 2.1, and update it whenever it changes.

In order to compute the visibility, we need the descriptions of the set of cells that the observer intersects, and for each cell, the set of lines bounding the cell in the arrangement of lines in the primal plane. Since computing the arrangement is too expensive, we take the following approach, which is similar to the method of Ghali and Stewart [2]. We compute the dual of S, the set of discontinuity lines, and the dual of s, the line segment observer. Let the duals be $S' = D(S)$ and $w_s = D(s)$. In the dual plane, each discontinuity line l is mapped to a point p_l and the segment s is mapped to a double wedges w_s (see Figure 2). Notice that segment endpoints A and B are mapped to two lines l_A and l_B, which are boundaries of w_s. The double wedge w_s divides the dual plane and the set of points S' into four sets S'_1, S'_2, S'_3 and S'_4, the first two sets are lying *outside*, and the other two sets are lying *inside* the wedge. When the dual of a discontinuity line is outside the wedge, it means that the line segment observer does not intersects the discontinuity line, and if the dual is inside the wedge, the observer intersects the discontinuity line. This observation helps us to easily see whether a discontinuity line crosses the observer or not.

Now let us see what happens when the visibility of the observer changes. As the observer moves in the plane, it crosses some visibility regions constructed by discontinuity lines. As long as the number and the order of these crossed regions do not change, the visibility of the observer will not change, and vice versa. So we need to show the conditions in which the crossed regions change.

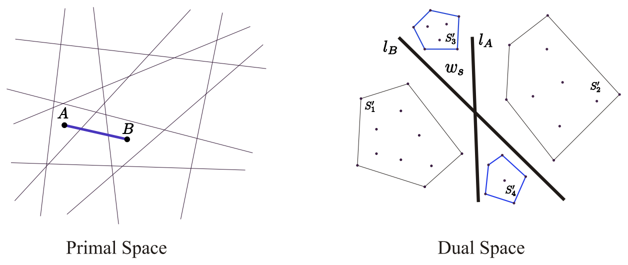


Figure 2: The figure on the left shows the arrangement of the discontinuity lines. The set of points shown on the right is the set of dual points of these lines. When the observer AB moves in the primal plane, its dual wedge w_s, also moves in the dual plane. Intersecting a new discontinuity line (respectively, stopping to intersect one) in the primal plane is signaled in the dual plane by boundaries of the wedge, l_A or l_B, intersecting one of the two convex hulls outside (respectively, inside) the wedge in dual plane.

A change in the set of crossed visibility regions is happening in two conditions: (i) the observer intersects a new discontinuity line, and so it enters a new region, or an already crossed discontinuity line no longer intersects the observer, and a visibility region will disappear (Figure 3), (ii) the order of intersection of two discontinuity lines will change, and an old region will disappear and a new one will appear (Figure 4). It can be easily seen that these two cases are the only situations that change the crossed regions. So, we need to consider these cases and update the visibility in each one.

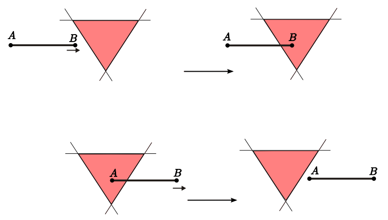


Figure 3: Events of type 1. In the figure above, the observer will intersect a new discontinuity line. The figure on the bottom shows that the observer stops intersecting a discontinuity line.

2.2.1 Event Type 1

As mentioned before, first type of events occurs in two cases: when the observer intersects a new discontinuity line, and it enters a new region, or when an already crossed discontinuity line no longer intersects the observer, and a visibility region will disappear (Figure

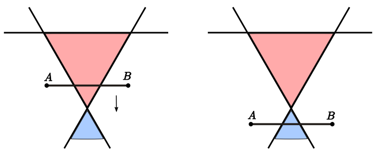


Figure 4: Event of type 2. The order of intersecting two discontinuity lines changes. A visibility region disappears and a new one appears.

3). When the observer intersects a new discontinuity line, it means that its dual point enters the double wedge. Similarly, when the observer stops to intersect a discontinuity line, it means that its dual point leaves the double wedge.

To detect these events, we only need to observe the dual lines of endpoints A and B. As the endpoints A (respectively B) gets closer to a line l on the boundary of a cell and becomes incident to it, the dual line $l_A = D(A)$ (respectively $l_B = D(B)$) becomes incident on the dual of that line $p_l = D(l)$. When A crosses l in the primal plane, the segment either enters another cell or leaves an already crossing cell. This is reflected in the dual plane by the point p_l crossing line l_A (see Figure 5). In what follows, we discuss how to update the structure upon insertion and deletion of points.

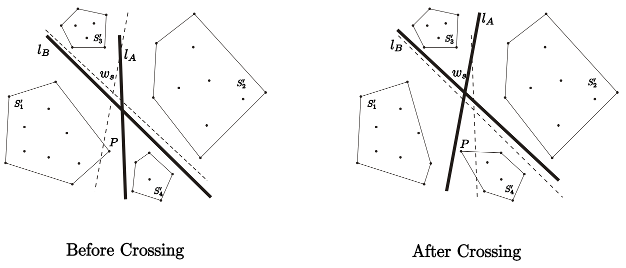


Figure 5: The figure on the left (respectively, right) shows the configuration in the dual plane before (respectively, after) a discontinuity line is crossed. In both figures, the solid lines are the duals of the current position of the observer endpoint A and B with the dashed lines showing the position of the duals after (respectively, before) crossing. As the discontinuity point P is crossed, P is deleted from one set of points and inserted into another one, while maintaining the convex hulls of both sets of points.

As long as the set of crossed discontinuity lines do not change, no event of type 1 will occur. If a new one is crossed or stops crossing, the data structure has to be updated. In the dual plane, these events reflected by a point entering or removing the wedge. In these cases, the point must be removed from one convex hull and inserted into anther one (one of the convex hulls is outside the wedge, and the other one is inside it). So, we need to maintain the convex hulls as points are added to or removed from them. To maintain a dynamic convex hull, a $O(\log^2 n)$ update were achieved early on [3], and it was improved to $O(\log n)$ in amortized [1].

The set of visible segments is not always updated when the segment observer crosses a discontinuity line, although the convex hulls must be updated to maintain the description of the current cell.

Consider the two convex hulls outside the wedge. When inserting a point into a convex hull in the dual plane, the point will be connected to the two common tangents between it and the hull. The points from the previous convex hull boundary that disappear from the new one correspond to the discontinuities in the primal plane that no longer bound the current cell in which an endpoint of observer exists. When deleting a point from a hull in the dual plane, an arbitrary number of points can appear on the hull boundary. These points correspond to the lines in the primal plane that bound the new cell which the viewpoint enters [2].

The two convex hulls that we maintain in the dual plane give us a concise description of the discontinuities in the primal plane which, when crossed, may require an update of the visibility cycle. The hulls can be stored in linear space with respect to the number of discontinuities. For every new positions of the endpoint A and B, the duals l_A and l_B are computed and a test performed to see if any point on either of the four convex hulls in the dual plane has switched sides.

In general, we have the following proposition:

Proposition 2 *After a preprocessing step in $O(m + n \log n)$ and computing the initial view of a line segment observer in $O((n + k) \log n)$ time, where m is the number of discontinuity lines and k is the discontinuities that initially intersect the line segment, the view from the observer among n line segment objects can be maintained in $O(\log n)$ each time a visibility change of type 1 occurs.*

2.2.2 Event Type 2

In the second type of visibility events, the order of intersection of two discontinuity lines changes, and an old region will disappear and a new one will appear (Figure 4). Notice that in this case, the set of crossed discontinuities will not change, and in dual plane, this means that the partition of points that the wedge has been created will not change.

Now let us see what happens in the dual plane upon occurring an event of type 2 (See Figure 6). As the observer AB come near to the intersection point of two discontinuity lines l_1 and l_2, the lines wp_{l_1} and wp_{l_2} in the dual plane come closer to each other, where w is the center of the wedge. At the event time, when AB passes through the intersection point of l_1 and l_2, wp_{l_1} and wp_{l_2} will lie on each other. After the event, the order of crossing these two discontinuity lines will change, and wp_{l_1} and wp_{l_2} will switch their polar order according to wedge center. It can be easily shown that each time two dual points p_{l_1} and p_{l_2} and wedge center w become collinear, an event of type 2 is occurred, and vice versa.

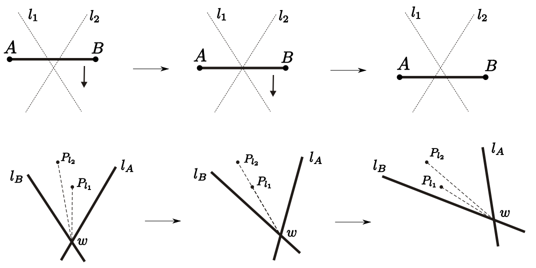


Figure 6: An event of type 2. As the observer AB approaches the intersection of discontinuity lines l_1 and l_2 (top), in dual plane, the lines $\overline{wp_{l_1}}$ and $\overline{wp_{l_2}}$ come closer to each other (bottom). At the event point (center) where AB passes through the intersection of l_1 and l_2, $\overline{wp_{l_1}}$ and $\overline{wp_{l_2}}$ will lie on each other. After the event, the order of crossing these two discontinuity lines will change, and in dual plane, $\overline{wp_{l_1}}$ and $\overline{wp_{l_2}}$ will switch their polar order according to wedge center (right).

Now we show how to handle this kind of event. We claim that this event can be handled with the algorithms for computing visibility of a moving viewpoint [4]. Assume that there are $m' = O(m) = O(n^2)$ points *inside* the wedge. Consider these m' points as endpoint objects and think of wedge center, w, as a moving viewpoint. When two of these endpoints become collinear with w, we can say that w has crossed the discontinuity line that passes through these two points.

With m' points, there will be $m'' = \Theta(m'^2) = O(n^4)$ discontinuity lines (notice that in average, m', the number of crossed discontinuities, is very smaller than $O(n^2)$). Using the algorithm of [4], we can handle events of type 2. As there are m'' discontinuity lines, we can compute the visibility complex of the scene in $O(m'' + m' \log m') = O(n^4 + n^2 \log n)$ time and $O(m') = O(n^2)$ working space. Having the visibility complex of the scene, each visibility event can be found and handled in $O(\log n)$ time. Each visibility event corresponds to a time that the wedge center w becomes collinear with two other points in the wedge, and an event of type 2 is occurred.

Two situations which are not considered yet are when an event of type 1 occurs, in which a new dual point will be added to the wedge, or when a point will remove the wedge. In these cases we must update the visibility complex of the points inside the wedge. When a new point enters the wedge, we have to insert this object in our visibility complex. This is done by computing a view around the point using the current visibility complex which can be done in $O(m' \log n)$ where m' is the number of points inside the wedge, using the techniques described by Rivière [4]. Similarly, when a point leaves the wedge, the edges of the visibility complex corresponding to segments passing through this point must be removed. This work can be done in $O(m')$.

In general, this approach leads to the following proposition:

Proposition 3 *After a preprocessing step in $O(m'^2 + (m' + n) \log n)$ and computing the initial view of a line segment observer in $O((m' + n) \log n)$, where m' is the number of discontinuity lines that intersect the segment observer, we can handle an event of type 1 in $O(\log n + m' \log n)$ time and an event of type 2 in $O(\log n)$ time. In other words, if there are k_1 events of type 1 and k_2 events of type 2, we can update the view in $O(k_1(\log n + m' \log n) + k_2 \log n)$.*

3 Conclusion

In this paper we defined topological visibility for a line segment observer among line segment objects in the plane. We proposed algorithms for computing the view in both static and dynamic cases.

In this work, we created the complete arrangement of all discontinuity lines. It would be better if we can use a local approach for the problem and avoid unnecessary computations. Another interesting problem is to scale upwards and introduce the problem in three dimensions. In the 3D case, there are some complications that make this a non-trivial task.

References

[1] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *In Proc. 43rd IEEE Sympos. Found. Comput. Sci*, pages 617–626, 2002.

[2] S. Ghali and A. J. Stewart. Maintenance of the set of segments visible from a moving viewpoint in two dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages V3–V4, 1996.

[3] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[4] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 421–423, 1997.

[5] G. Vegter. Computational topology. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 32. CRC Press LLC, Boca Raton, FL, 2th edition, 2004.

Computing Direct Shadows Cast by Convex Polyhedra

Julien Demouth[*] Xavier Goaoc[†]

Abstract

We present an exact method to compute the boundaries between umbra, penumbra and full-light regions cast on a plane by a set of disjoint convex polyhedra, some of which are light sources. This method builds on a recent characterization of topological visual event surfaces presented in a companion paper.

1 Introduction

Shadows play an important role in our understanding of three-dimensional shapes from two-dimensional pictures [18, 23]. As such, their realistic rendering has been a central problem in Computer Graphics for the last decades [12]. While illumination simulation methods such as photon mapping [16] now produce excellent results, the boundaries of shadows are obtained *a posteriori* to the illumination simulation. It is sometimes desirable to determine shadow boundaries before that phase, that is, to deduce it from the geometry of the scene. This is the case, for instance, when using shadow volumes [1, 5] for real-time rendering or discontinuity mesh [15, 17] to improve radiosity methods [4, 20]. Unlike hard shadows cast by punctual light sources, whose geometry is well understood [25], soft shadows cast by extended light sources still prove elusive. The classical approach traces shadow boundaries back to *visual event surface* [22], that is points from which the view of the scene changes. The classical visual event surfaces [13], however, capture much more than direct shadows, for example shadows cast by indirect lighting. As a result, methods based on them such as the *discontinuity mesh* [9, 15, 17, 21] or the *visibility skeleton* [10, 11] produce sets of curves that are too large to be used in practice. In this abstract, we present a method that computes exactly the shadows cast by a set of disjoint convex objects, some of which are light sources, ignoring the effects of light reflection which are arguably less important (see, for example, [4]), and evaluate its performance on a few types of random scenes.

[*]Dept. of Computer Sci. & Eng., POSTECH, Pohang, Korea, julien@postech.ac.kr

[†]VEGAS Project, INRIA Grand Est, Nancy, France, Xavier.Goaoc@loria.fr

2 Characterization of shadow boundaries

In a companion paper [8], we studied visual events related to changes in the topology of the apparent contour of a collection of convex sets. Here, we first show that these events are sufficient to capture direct shadows. We then associate to each point of a visual event surface a combinatorial structure, its *germ*, which characterizes the type of change occurring at that point. We show that only points with certain germs can participate in each type of shadow boundary, and that these germs can be associated with the arcs of a graph encoding the visual event surfaces, the visibility skeleton.

Topological visual events. Visual event surfaces are defined as the locus of points at which the view of the scene undergoes a qualitative change. The notions of "view" and "qualitative change" need to be defined carefully, though, as different definitions capture different phenomena. For polyhedral scenes, the view is usually defined as the (abstract) graph obtained by projecting the visible portions of the polyhedra's edges, each element of the graph being labelled by its three-dimensional generators, and two views are qualitatively equivalent if they are isomorphic, as labelled graphs [13]. The corresponding visual event surfaces are of two types [13]: EV surfaces, consisting of origins of rays through a vertex and an edge, and EEE surfaces, formed by origins of rays through three edges (see Figure 1). Thus, the visual event surfaces of a

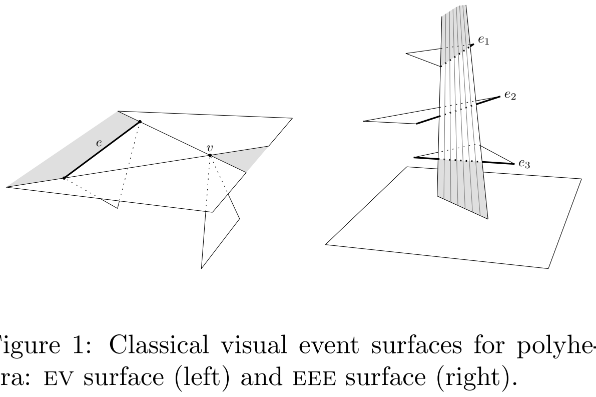


Figure 1: Classical visual event surfaces for polyhedra: EV surface (left) and EEE surface (right).

scene consisting of a single convex polyhedron are the planes supporting its faces. Yet, since the polyhedron is totally visible from anywhere in space, it induces no shadow boundary at all.

These choices of "view" and "qualitative change" are sensitive to the changes in the view not only of the polyhedra, but also of their meshing: events occur whenever a facet starts or ceases to be visible. In a companion paper [8], we studied the visual events corresponding to the topological changes in the apparent contour of a collection of disjoint convex sets. Specifically, we define the view from a point as the set of directions of rays from that point that meet the scene tangentially and declare two such views equivalent if there exists an homeomorphism from the space of direction into itself that maps one view into the other. The associated "topological" visual event surfaces suffice to capture direct shadows.

Theorem 1 *The boundary of the direct shadow cast by a collection of disjoint convex objects is contained in the trace of visual event surfaces corresponding to topological changes in the apparent contour.*

Call a ray *tritangent* if it sees tangentially three objects and *limiting bitangent* if it sees tangentially two objects and lies in a plane tangent to both of them. In the case of disjoint convex polyhedra, we show [8, Theorem 3] that there are two types of such visual event surfaces: EEE surfaces where the three edges come from distinct polyhedra, that is origins of tritangent rays, and EV surfaces where the edge and the vertex come from distinct polyhedra and span a plane tangent to both, that is, origins of limiting bitangent rays.

Germ of a ray. Let ρ be a tritangent or limiting bitangent ray. Up to symmetries, the projection of the the objects seen tangentially by ρ are locally equivalent (that is, homeomorphic) to one of eight basic situations (see Figure 2) which we call the *germ* of ρ; we group the germs into four *types*, from 1 to 4. A ray has one germ for each triple (resp. pair) of object to which it is tritangent (resp. limiting bitangent). Points that belong to a umbra/penumbra or a penumbra/full light boundary are origins of rays with prescribed germ types:

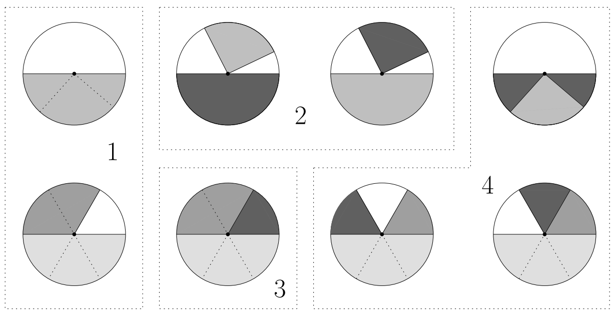


Figure 2: The germs of EV surfaces (top) and EEE surfaces (bottom). Sectors covered by the projection of an object are shaded in grey, lighter colors corresponding to objects closer to the origin of the ray. The numbers $1, \ldots, 4$ indicate the type of the germ(s) of the corresponding box.

Theorem 2 *A point belongs to the boundary between the umbra and the penumbra (resp. penumbra and full light) regions cast by a collection of convex sets only if it sees a light source tangentially along a ray with germ of type* 1 *(resp.* 2*) or if it sees a light source along a ray with germ of type* 3.

Visibility skeleton. Let $\mathcal{C}$ denote a set of convex polyhedra and $A(\mathcal{C})$ the arrangement, in ray space, of the sets of tritangent and limiting bitangent rays. $A(\mathcal{C})$ encodes the visual event surfaces of $\mathcal{C}$ in the sense that these surfaces are the origins of the rays that make up the faces of $A(\mathcal{C})$. Two rays $\rho \subset \rho'$ such that the segment joining their origin does not intersect $\mathcal{C}$ belong to the same cell in $A(\mathcal{C})$. We can thus "collapse" $A(\mathcal{C})$ by identifying all such rays. One way to do that is to consider the quotient space $\mathcal{R}/\sim$ where $\mathcal{R}$ is the space of rays and:

$$(p+\mathbb{R}_+\vec{u}) \sim (q+\mathbb{R}_+\vec{v}) \Leftrightarrow \vec{u} = \vec{v} = \pm\overrightarrow{pq} \text{ and } [pq]\cap\mathcal{C} = \emptyset.$$

The origins of all rays in a given equivalence class make up a *maximal free segment*: a segment that does not intersect $\mathcal{C}$ and is maximal for this property with respect to the inclusion[1]. There is thus a natural bijection between $\mathcal{R}/\sim$ and the space of maximal free segments[2]; we call the image of $A(\mathcal{C})$ under this bijection the *visibility skeleton* of $\mathcal{C}$ relative to topological changes in the apparent contour (by analogy to the classical visibility skeleton [11] defined to encode classical visual event surfaces). For a collection of disjoint convex polyhedra in generic position[3], this visibility skeleton is a graph whose arcs are maximal free segments supported by tritangent or limiting bitangent rays to a given triple/pair of objects, and whose nodes are maximal free segments supported by a line through two vertices (VV node), one vertex and two edges (VEE node) or four edges (EEEE node).

Lemma 3 *Two rays in the same facet of $A(\mathcal{C})$ have the same germ.*

As a consequence, all rays corresponding to a given arc of the visibility skeleton have the same germ, which we call the *germ of the arc.* The characterization of Theorem 2 then extend to arcs of the visibility skeleton, and thus to visual event surfaces.

[1]Note that this differs from the usual definition in that a maximal free segment is not allowed to be tangent to an object.

[2]This bijection is not an homeomorphism as the natural topologies on these spaces differ, but this has no consequence for our purposes.

[3]The same holds for non-generic position but additional care in the definition is required.

3 Algorithm outline

Our algorithm takes as input a set $\mathcal{C}$ of disjoint convex polyhedra and a plane Π, and consists of 3 steps:

1. Compute the visibility skeleton of $\mathcal{C}$ and the germs of its arcs.
2. Compute the arrangements, in the plane Π, of the curves traced by the surfaces swept by the relevant arcs of the skeleton. We compute two arrangements: (a) with arcs of type 1 tangent to a light source in last position and arcs of type 3 with an endpoint on a light source, and (b) one with arcs of type 2 tangent to a light source.
3. At each curve crossing, remove the arcs not on the shadow boundary locally (see Figure 3).

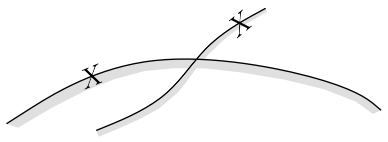


Figure 3: Two arcs incident to a crossing of two curves in arrangement (a) do not belong to the shadow boundary (the shading indicates on which side of the arc the source becomes locally invisible). A similar criterion holds for crossings in arrangement (b).

The output consists of two collections of closed non-crossing loops which, by Theorem 2, contain the boundaries between, respectively, umbra and penumbra – for (a) – and penumbra and full light – for (b). Testing if a given loop is an actual shadow boundary can be done by computing the view of a single point. The visibility skeleton of k convex polyhedra of total complexity n has size $\Theta(n^2k^2)$ [3] and can be computed in time $O(n^2k^2 \log n)$ by a sweep-plane algorithm [14, Chapter 8]. The complexity of the arrangements (a) and (b) is $O(n^3k^3)$ [7], so altogether the algorithm runs in $O(n^3k^3 \log n)$ time in the worst case.

4 Implementation and experimental results

Our implementation was done in `C++` using exact number types and geometrical objects from `CGAL` library. The implementation assumes the objects are in general position in the sense that no 3 vertices are collinear, no 4 vertices (from distinct polyhedra) coplanar and no 10 vertices on a common quadric surface. For Step 1, we compute the VV nodes by brute-force enumeration and use an implementation by Zhang et al. [26] to compute the VEE and EEEE nodes; we retrieve the connectivity using local computations. For Step 2, we use the 2D arrangement package from `CGAL` [24], the arcs of conic curves induced by EEE arcs being described by a rational equation and two algebraic endpoints[4]. See [6, Chapter 6] for a more detailed discussion on the implementation.

We tested three different types of random scenes. In each case, we chose a random collection of k disjoint unit spheres, and for each pick p points on its surface and compute their convex hull. The three types of distribution of centers of spheres that we consider are:

(i) k centers randomly distributed inside a cube,

(ii) k centers randomly distributed on a constant number of horizontal slices of a cube,

(iii) $k-1$ centers randomly distributed on a constant number of horizontal slices of a cone with apex the k^{th} sphere, which is the light source.

We run tests for $k = 10, 30$ and 50 and p ranging between 4 and the greatest power of 2 manageable. Disjointedness is ensured by throwing away centers that violate it and generating them again. For each value of k and p we created 10 scenes for type (i) and a single one for types (ii) and (iii). We analyzed our method using three indicators:

- $gain_1 = \frac{b-a}{b}$, where a denotes the size (number of nodes) of our visibility skeleton and b that of the classical visibility skeleton [11]. It measures the gain obtained by considering "topological" visual events rather than the classical visual events.
- $gain_2 = \frac{d-c}{d}$, where d denotes the number of arcs of our visibility skeleton and c the number of arcs used in Step 2. It measures the effectiveness of the filter based on germ conditions.
- $gain_3 = \frac{e-f}{e}$, where e denotes the complexity of the arrangements (a) and (b) and f that of the output of Step 3. It measures the reduction in size obtained by storing only the (candidate) contours of direct shadows (versus keeping direct discontinuities in the interior of the penumbra).

Our measures show that all three gains are high, most of the time. Specifically,

- $gain_1$ increases with p, reaching 50% for $p = 16$ and 75% for $p = 64$, independently of k. Since the visibility skeleton has a rather large size, this gain is significant; for instance, for $k = 30$ and $p = 128$, $a = 290000$ and $b = 50000$.
- $gain_2$ is above 85% for all tested scenes, and increases with k.
- $gain_3$ is below 10% for the limit between penumbra and full-light, but above 70% for the limit between umbra and penumbra.

[4]Note that a polygonal approximation of these arcs is performed at visualization.

A comparison of some outputs of our program with images produced using the PBRT ray-shooting program [19] also suggests that the results are visually convincing. Due to space limitation, we refer to [6, Chapter 6] for a detailed account of the performances.

5 Conclusion

Our experiments suggest that the method we propose is simple enough to be put in practice and leads to significant gains over the classical approach. Of course, these experiments are on small-scale ($\leq 10^4$ triangles) non-realistic scenes. The main challenges to overcome these limitations include the efficient computation of EEEE nodes and the understanding of "topological" visual events in non-convex settings. A first progress on the latter was recently made by Batog [2]. Our results also suggest that the boundary of shadow regions is simple enough to be of interest in Computer Graphics.

References

[1] U. Assarsson and T. Akenine-Möller. A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware. *ACM Transactions on Graphics*, 22(3):511–520, 2003.

[2] G. Batog. *Événements visuels de polyèdres*, Master's thesis, MPRI, ENS Cachan, 2008.

[3] H. Brönnimann, O. Devillers, V. Dujmović, H. Everett, M. Glisse, X. Goaoc, S. Lazard, H.-S. Na and S. Whitesides. Lines and Free Line Segments Tangent to Arbitrary Three-dimensional Convex Polyhedra. *SIAM Journal of Computing*, 37(2):522–551, 2007.

[4] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press, 1993.

[5] F. C. Crow. Shadow Algorithms for Computer Graphics. *Computer Graphics*, 11(2):242–248, 1977.

[6] J. Demouth. *Événements visuels de convexes et limites d'ombres.* PhD thesis, Université Nancy 2, 2008.

[7] J. Demouth, O. Devillers, H. Everett, M. Glisse, S. Lazard and R. Seidel. On the Complexity of Umbra and Penumbra. *Computational Geometry: Theory and Applications*, to appear, 2009.

[8] J. Demouth and X. Goaoc. Topological Changes in the Apparent Contour of Convex Sets. Manuscript, 2009.

[9] G. Drettakis and E. Fiume. A Fast Shadow Algorithm for Area Light Sources Using Back-projection. *Proceedings of ACM* SIGGRAPH *94*, pages 223–230, 1994.

[10] F. Duguet and G. Drettakis. Robust Epsilon Visibility. *Proceedings of ACM SIGGRAPH 2002*, pages 567–575, 2002.

[11] F. Durand, G. Drettakis and C. Puech. The Visibility Skeleton: a Powerful and Efficient Multi-Purpose Global Visibility Tool. *Proceedings of ACM* SIGGRAPH *97*, pages 89–100, 1997.

[12] P. Dutré and K. Bala and P. Bekaert. *Advanced Global Illumination.* A.K. Peters, 2006.

[13] Z. Gigus and J. Malik. Computing the Aspect Graph for Line Drawings of Polyhedral Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):113–122, 1990.

[14] X. Goaoc. *Structures de visibilité globales : taille, calcul et dégénérescences.* PhD thesis, Université Nancy 2, 2004.

[15] P. S. Heckbert. Discontinuity Meshing for Radiosity. *Proceedings of the Eurographics Workshop on Rendering*, pages 203–215, 1992.

[16] H. W. Jensen. Global Illumination Using Photon Maps. *Proceedings of the Eurographics Workshop on Rendering*, pages 21–30, 1996.

[17] D. Lischinski, F. Tampieri and D. P. Greenberg. A Discontinuity Meshing Algorithm for Accurate Radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, 1992.

[18] P. Mamassian, D. C. Knill and D. Kersten. The Perception of Cast Shadows. *Trends in Cognitive Sciences*, 2(8):288–295, 1998.

[19] M. Pharr and G. Humphreys. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann, 2004.

[20] F. Sillion and C. Puech. *Radiosity and Global Illumination.* Morgan Kaufmann, 1994.

[21] A. J. Stewart and S. Ghali. Fast Computation of Shadow Boundaries Using Spatial Coherence and Backprojections. *Proceedings of ACM* SIGGRAPH *94*, pages 231–238, 1994.

[22] S. J. Teller. Computing the Antipenumbra of an Area Light Source. *Proceedings of ACM SIGGRAPH 92*, pages 139–148, 1992.

[23] L. R. Wanger. The Effect of Shadow Quality on the Perception of Spatial Relationships in Computer Generated Images. *Computer Graphics*, 25(2):39–42, 1992.

[24] R. Wein, E. Fogel, B. Zukerman and D. Halperin. 2D Arrangements. CGAL User and Reference Manual, Release 3.3, 2007. http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html#Pkg:Arrangement2.

[25] A. Woo, P. Poulin and A. Fournier. A Survey of Shadow Algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, 1990.

[26] L. Zhang, H. Everett, S. Lazard, C. Weibel and S. Whitesides. On the Size of the 3D Visibility Skeleton: Experimental Results. *Proceedings of the Sixteenth Annual European Symposium on Algorithms (ESA 2008)*, to appear, 2008.

Straight Walk Algorithm Modification for Point Location in a Triangulation [*]

Roman Soukal [†] Ivana Kolingerová [‡]

Abstract

Finding an element in a mesh which contains a query point is a very frequent task in computational geometry. This paper shows a modification of one planar technique for point location in triangulation. This algorithm is based on a modification of the straight walk location algorithm combined with the visibility walk algorithm where straight walk brings speedup and visibility walk reliably finds a target. This modification does not improve the worst-case running time although empirical data shows that its performance is better than previous *Straight walk* algorithm.

1 Introduction

For a query point q and a given triangulation T of n vertices in the plane the point location problem usually means how to find a triangle ω from T which contains the query point. One of the possibilities how to solve this problem is to use one of the walking location techniques. The name of these algorithms has arisen from the way of locating the triangle ω which contains q. For a starting triangle α chosen as one of the triangles of T and the query point q the walking strategy makes use of connectivity in the triangle mesh and it goes through triangles between α and ω. There are three main types of walking strategies. First, the visibility walk makes use of an orientation edge test to determinate which triangle is the next. Second, the straight walk passes all triangles in the mesh between α and ω which are intersected by a line pq where p is a point inside α. Finally, the orthogonal walk passes all triangles in the mesh between α and ω in the directions of coordinate axes. The triangle α may be chosen randomly or as the closest triangle to q from the set A of randomly chosen triangles from T, $\|A\| \ll \|T\|$ [7]. This choice improves speed of the algorithm. Walking algorithms are frequently used despite suboptimal complexity ($O(\sqrt[3]{n})$ up to $O(\sqrt{n})$ [3] - depending on how the α was chosen).

Sophisticated data structures such as DAG [1], [6], skip list [11], quad tree, buckets [10], uniform grid [9], [12] and data structures based on random sampling [8], [2] are used in the point location algorithms with the best known complexity $O(\log n)$ per point query. However, these algorithms have some disadvantages. First, these data structures consume generally $O(n)$ amount of memory which may be a problem for huge datasets. Second, implementation effort for most of these structures may be nontrivial (especially for modifications of these structures). Finally, most of these structures are hierarchical and the top level of the hierarchy may be a bottleneck in case of parallelization. Walking algorithms do not need any extra memory, their implementation is rather simple and their usability for parallelization is good, thus often they are a better choice than optimal time complexity solutions.

This paper shows a straight walk algorithm modification which is faster than the standard straight walk algorithm [3]. The paper is organized as follows. Section 2 and 3 present already existing algorithms *Remembering Stochastic walk* and *Straight walk*. Section 4 shows our modification of this algorithm, problem resulting from the character of this modification and its solution. Empirical results are presented in Section 5.

2 Remembering Stochastic Walk

Visibility walk algorithms use orientation test for a triple of points t, u, v (Equation 1).

$$orientation(t,u,v) = sgn\left(\begin{vmatrix} u_x - t_x & v_x - t_x \\ u_y - t_y & v_y - t_y \end{vmatrix}\right) \quad (1)$$

Decision, whether or not the edge rl of the current triangle τ ($\tau = srl$) should be crossed to continue the walk into the next triangle depends on the result after substitution q, r, l to the Equation 1 where q is the query point. The walk continues to the next triangle over the edge rl if $orientation(q,r,l) = -orientation(s,r,l)$ where s is vertex of τ, $r, l \neq s$ and q is the query point. Simple visibility walk algorithms use edges of τ for tests in the given order, but these visibility walks may loop for a non-Delaunay triangulation [4]. For non-Delaunay triangulation it is necessary to choose the tested edges of τ in a random order. This modification is called *Stochastic*. Naturally it is not necessary to test the edge incident with

[*]This work is supported by the Grant Agency of the Czech Republic - the project GA 201/09/0097

[†]Department of Computer Science, Faculty of Applied Sciences, University of West Bohemia, soukal@kiv.zcu.cz

[‡]Department of Computer Science, Faculty of Applied Sciences, University of West Bohemia, kolinger@kiv.zcu.cz

the previous triangle in the walk. This improvement is called *Remembering* and may save up to one orientation test for each triangle. Therefore, one or two orientation tests are needed for each triangle (except the triangle α).

3 Straight Walk

Generally, the straight walk passes all triangles in the triangulation T intersected by the line pq where q is the query point and p is a point in the given starting triangle α. The standard straight walk algorithm [3] is composed of two steps and point p is chosen as a vertex of α. In the initialization step (grey color in Figure 1) a triangle γ incident to p and intersected by pq must be found. During this step one orientation test is needed for each visited triangle and the number of visited triangles is at most the degree of p, thus at most n.

After the initialization is completed, the straight walk (black color in Figure 1) may start. For each triangle τ, $\tau = srl$ in the straight walk, the line pq goes into $\tau, \tau \neq \alpha$ through edge $e = rl$ (see τ in Figure 1). Depending on the *orientation*(s, p, q), new r (or new l) is selected as s. The singular case *orientation*$(s, p, q) = 0$ is added to one of these situations. Now the straight walk goes out of τ through the new edge $e = rl$. By testing on which side of e the q lies it is decided whether the τ contains the q or whether the walk must go on. In the latter case the walk goes to the neighbor of τ through e. Therefore the *Straight walk* evidently needs two orientation tests per triangle and some orientation tests in the initialization step. Pseudo code of the algorithm is given as Algorithm 1 [3].

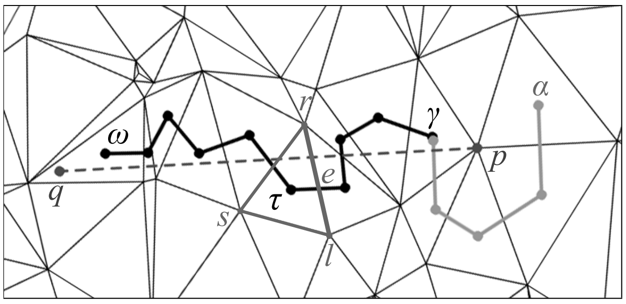


Figure 1: Straight walk example

4 The Proposed Modification of Straight Walk

The first idea is to simplify the initialization step of the algorithm in Section 3 to a constant number of operations. The second idea is to use a cheaper operation than the orientation test used in *Straight walk* algorithm in Section 3.

```
// traverses the triangulation $T$
Input:
  • the query point $q$
  • the chosen starting triangle $\alpha, \alpha \in T$
Output:
  • the triangle $\omega$ which contains $q$
// following the line segment from $p$ to $q$
// initialization step
$p$ = vertex of $\alpha$;
if $orientation(r, p, q) < 0$ then
    while $orientation(l, p, q) < 0$ do
        $r$ = $l$;
        $\tau$ = neighbor of $\tau$ trough $pl$;
        $l$ = vertex of $\tau, l \neq p, l \neq r$;
    end
else
    repeat
        $l$ = $r$;
        $\tau$ = neighbor of $\tau$ trough $pr$;
        $r$ = vertex of $\tau, r \neq p, r \neq l$;
    until $orientation(r, p, q) < 0$ ;
end
// end of initialization and start of straight walk
// now $pq$ has $r$ on the right and $l$ on the left side
while $orientation(q, r, l) < 0$ do
    $\tau$ = neighbor of $\tau$ trough $rl$;
    $s$ = vertex of $\tau, s \neq r, s \neq l$;
    if $orientation(s, p, q) < 0$ then
        $r$ = $s$;
    else
        $l$ = $s$;
    end
end
// end of Straight walk step
// now $\tau$ contains $q$
return $\tau$;
```

Algorithm 1: Straight Walk

A fundamental prerequisite for the initialization step is to suitably choose the point p. In Section 3, p is chosen as one vertex of the starting triangle α but it is not necessary. The main idea is to choose p in a way that no other operations in the initialization step are needed. First, the point s is chosen as the closest vertex from α to q. The edge $e = rl$ is the edge of α and $\alpha = srl$. Next, p is chosen on e where $r, l \neq p$ and $\|pq\| > 0$. Now the straight walk may start. For each triangle τ in the straight walk (except α) the edge e of τ is edge used to cross to τ and s is the vertex of τ facing e. The line pq goes out of τ through the edge e that is determined by the means of using the orientation test. If s lies on the left side of pq, e $(s \in e)$ is the edge of τ on the right of s, else e $(s \in e)$ is the edge of τ on the left of s. By testing on which side of e the q lies it is decided whether τ contains q or whether the walk must go on.

For each triangle τ the position of s is tested against the line pq and only s is changing during the walk, it is therefore possible to use an implicit line equation test in the place of the orientation test to speed up the process. The implicit line equation of pq is computed in the initialization step (Equations 2, 3, 4) and the position of s is found by a substitution into Equation 5.

$$\lambda : a \cdot x + b \cdot y + c = 0 \tag{2}$$

$$[a, b, c] = [p_x, p_y, 1] \times [q_x, q_y, 1] \tag{3}$$

$$a = p_y - q_y, b = q_x - p_x, c = p_x \cdot q_y - p_y \cdot q_x \tag{4}$$

$$position(\lambda, s) = sgn(a \cdot s_x + b \cdot s_y + c) \tag{5}$$

The implicit line equation test is subject to higher numerical imprecision than the orientation test but the straight walk algorithm is robust enough to resist it. Now there is one orientation test and one position test per triangle. The orientation test is used to detect whether the triangle ω was found. A direct replacement of this orientation test is problematic because two points are changing during the walk, therefore the original algorithm must be modified. For this modification a new line $\lambda_n, \lambda_n \perp pq, q \in \lambda_n$ must be computed in the initialization step (Equation 6, 7).

$$\lambda_n : a \cdot x + b \cdot y + c = 0 \tag{6}$$

$$a = \lambda_y, b = \lambda_x, c = \lambda_x \cdot q_y - \lambda_y \cdot q_x \tag{7}$$

The orientation test for detection, whether the triangle ω was found, is replaced with the position test of λ_n and s. If s lays on the other side of λ_n than p, the straight walk ends. A situation is possible where $\tau \neq \omega$ at the end of the straight walk (see Figure 2). Because of it, *Remembering Stochastic walk* algorithm is used for final location. As a rule, this final location is very short, but extreme cases exist where almost the whole walk is performed by *Remembering Stochastic walk*. However, this situation is not probable and degradation of *Straight walk* to *Remembering Stochastic walk* is not a problem, because *Remembering Stochastic walk* is not dramatically worse (see Section 5). It is possible to a find more detailed description of this modified variant of *Straight walk* in the pseudo code Algorithm 2 and in the example Figure 2. The straight walk step is coloured black and the final location by *Remembering Stochastic walk* is grey. The triangle γ is the triangle where the straight walk step ends and *Remembering Stochastic walk* starts, the edge e_γ is the edge to γ and the point s_γ is the point s for γ, whose position at the other side of λ_n than p causes the end of the straight walk.

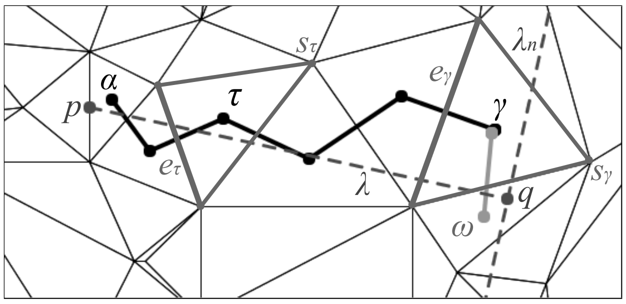


Figure 2: Modified Straight walk example

```
// traverses the triangulation $T$
**Input**:
  • the query point $q$
  • the chosen start triangle $\alpha, \alpha \in T$
**Output**:
  • the triangle $\omega$ which contains $q$
// initialization step
$\tau$ = $\alpha$;
$s$ = the vertex of $\tau$ closest to $q$;
$e$ = $rl$ = edge of $\tau$ facing $s$;
choose $p$:  $p \in e,\ r, l \neq p, \|pq\| > 0$;
compute the line $\lambda$ ($pq$);
compute the line $\lambda_n$;
$found$ = false;
// end of initialization and start of straight walk
repeat
    if $position(\lambda_n, s) > 0$ then
        $found$ = true;
    else
        if $position(\lambda, s) > 0$ then
            $e$ = edge of $\tau$ on the left from $s, s \in e$;
        else
            $e$ = edge of $\tau$ on the right from $s, s \in e$;
        end
        $t$ = neighbor of $\tau$ trough $e$;
        $s$ = vertex of $\tau$ facing $e$;
    end
until $found$ ;
// end of the straight walk step
// final location
$\tau$ = Remembering Stochastic Walk ($\tau, q$);
// now $\tau$ contains $q$
return $\tau$;
```

Algorithm 2: Modified Straight Walk

5 Empirical Results

The following algorithms were tested: *Remembering walk*, *Remembering Stochastic walk*, *Straight walk* (Algorithm 1) and *Modified Straight walk* (Algorithm 2). Tests were performed on Delaunay triangulations of different sizes of datasets (especially 100000 and 1000000 points) and on different types of point distributions (randomly in square, grid, gauss distribution, clusters, arcs, real data). First triangle δ was randomly chosen for each location. 1000000 of randomly generated points were located by each algorithm on each dataset. Selected results are in Table 1. The following properties were examined for each algorithm: the average length of the walk (#Δ), the average number of the orientation tests (#ori), the average number of position tests (#pos) and the average time (t[μs])per one location (tested on Intel Q6600 2,40GHz). The algorithms are coded in Java with the double precision floating point arithmetic.

Algorithm	#Δ	#ori	#pos	t[μs]
	ϕ per located point			
Delaunay triangulation of 100.000 points distributed in a square				
Rem. walk	377.80	508.55	0	43.97
Rem. Stoch. walk	360.12	465.11	0	58.04
Straight walk [3]	338.98	677.21	0	57.14
Mod. Straight walk	340.83	3.77	677.23	41.07
Delaunay triangulation of 1.000.000 points distributed in a square				
Rem. walk	1185.93	1590.30	0	214.97
Rem. Stoch. walk	1122.87	1443.80	0	255.57
Straight walk [3]	1069.09	2137.46	0	234.16
Mod. Straight walk	1071.11	3.76	2138.37	188.61
Delaunay triangulation of 100.000 points distributed in a clusters				
Rem. walk	214.25	290.09	0	16.73
Rem. Stoch. walk	215.06	279.95	0	24.33
Straight walk [3]	173.66	346.68	0	20.87
Mod. Straight walk	174.40	5.21	341.42	12.06
Delaunay triangulation of 70.433 points from real data				
Rem. walk	324.00	437.51	0	34.47
Rem. Stoch. walk	324.42	422.14	0	47.76
Straight walk [3]	297.08	593.46	0	46.81
Mod. Straight walk	298.02	5.22	588.59	32.6

Table 1: Comparison of algorithms

With regard to the number of tests, *Remembering Stochastic walk* with about 1.30 orientation tests per triangle is the best. Non-Stochastic *Remembering walk* is a little worse with about 1.35 orientation tests per triangle. Both straight walks have about two tests per triangle (a small difference is due to the initialization step (the standard *Straight walk*) or due to the final location (*Modified Straight walk*)) but *Modified Straight walk* mainly uses faster tests. With regards to the number of visited triangles, the best are the *Straight walk* algorithms. In comparison of time per one location, *Modified Straight walk* is the fastest way because of faster tests. *Remembering walk* has good results too but it is only usable for Delaunay triangulations. *Remembering Stochastic walk* has very good results regarding the number of tests per triangle but the cost for the randomization is too high, therefore the standard *Straight walk* with two orientation tests per triangle is a little faster.

Dataset	ϕ #Δ	*max* #Δ
	per located point	
DT of 100.000 random pts	1.77	10
DT of 1.000.000 random pts	1.76	9
DT of 100.000 pts in clusters	3.23	341
DT of 70.433 pts from real data	3.23	186

Table 2: Length of final location by Remembering Stochastic walk in Modified Straight walk algorithm

Experiments show that the length of the final location with *Remembering Stochastic walk* in *Modified Straight walk* is generally short (see Table 2). There are fluctuations on some types of datasets (e. g. clusters) but they are rare and may be ignored on average.

6 Conclusion

The proposed algorithm was verified to find a point in a Delaunay triangulations in a better time than the popular *Remembering Stochastic walk*. Our algorithm can be used also for non-Delaunay triangulations. Testing on such data is our future work.

Acknowledgments

We would like to thank Mr. Brož and Mr. Hlaváček for their feedback and inspiring discusion.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. *Computational Geometry, Algorithms and Applications.* Berlin Heidelberg: Springer, 1997.

[2] O. Devillers. *Improved Incremental Randomized Delaunay Triangulation* Proceedings of the 14th Annual Symposium on Computational Geometry 1998, 106-115, 2001.

[3] O. Devillers, S. Pion, M. Teillaud. *Walking in a Triangulation.* Proceedings of the 17th Annual Symposium on Computational Geometry 2001, 106-114, 2001.

[4] L. De Floriani, B. Falcidieno, G. Nagy, C. Pienovi. *On Sorting Triangles in a Delaunay Tesselation.* Algorithmica 6, 522-532, 1991.

[5] I. Kolingerová. *A Small Improvement in the Walking Algorithm for Point Location in a Triangulation.* 22nd European Workshop on Computational Geometry, 221-224, 2006.

[6] I. Kolingerová, B. Žalik. *Improvements to Randomized Incremental Delaunay Insertion.* Computers & Graphics, 26, 477-490, 2002.

[7] E. P. Mücke, I. Saias, B. Zhu. *Fast Randomized Point Location without Preprocessing in Two- and Three-dimensional Delaunay Triangulations.* Proceedings of the 12th Annual Symposium on Computational Geometry 1996, 274-283, 1996.

[8] K. Mulmuley. *Randomized Multidimensional Search Trees: Dynamic Sampling.* Proceedings of the 7th Annual Symposium on Computational Geometry 1991, 121-131, 1991.

[9] S. W. Sloan. *A Fast Algorithm for Constructing Delaunay Triangulations in the Plane.* Advanced Engineering Software, 9(1):34-55, 1987.

[10] P. Su, R. L. S. Drysdale. *A Comparison of Sequential Delaunay Triangulation Algorithms.* Proceedings of the 11th Annual Symposium on Computational Geometry 1995, 61-70, 1995.

[11] M. Zadravec, B. Žalik. *An Almost Distribution Independent Incremental Delaunay Triangulation Algorithm.* The Visual Computer, 21(6):384-396, 2005.

[12] B. Žalik, I. Kolingerová. *An Incremental Construction Algorithm for Delaunay Triangulation Using the Nearest-point Paradigm.* International Journal of Geographical Information Science, 17(2):119-138, 2003.

Every 4-connected Möbius triangulation is geometrically realizable

Atsuhiro Nakamoto[*] Shoichi Tsuchiya[†]

Abstract

Let M be a map on a surface F^2. A *geometric realization* of M is an embedding of F^2 into a Euclidian 3-space $\mathbb{R}^3$ with no self-intersection such that each face of M is a flat polygon. In earlier researches, it has been proved that every triangulation G on the projective plane has a face f such that the triangulation $G - f$ on the Möbius band obtained from G by removing the interior of f has a geometric realization. In this paper, we shall characterize such a face f in a triangulation G on the projective plane. This immediately implies that every 4-connected triangulation on a Möbius band has a geometric realization.

1 Introduction

A *map* is a fixed embedding of a graph on a surface F^2. A *triangulation* on a surface F^2 is a map on F^2 such that each face is bounded by a 3-cycle, where a k-cycle means a cycle of length k. We suppose that the graph of a map is always *simple*, i.e., with no multiple edges and no loops.

Let M be a map on a surface F^2. A *geometric realization* of M is an embedding of F^2 into a Euclidian 3-space $\mathbb{R}^3$ with no self-intersection such that each face of M is a flat polygon.

Steinitz's proved that a spherical map G has a geometric realization if its graph is 3-connected [7]. (He actually proved that G has a geometric realization such that no two adjacent faces lie on the same plane in $\mathbb{R}^3$ if and only if G is 3-connected.) Moreover, Archdeacon et al. proved that every toroidal triangulation has a geometric realization [1]. In general, Grünbaum conjectured that every triangulation on any orientable closed surface has a geometric realization [6], but Bokowski et al. showed that a triangulation by the complete graph K_{12} with twelve vertices on the orientable closed surface of genus 6 has no geometric realization [3]. Hence Grümbaum's conjecture is no longer true now but it is still open for the orientable closed surface of genus from 2 to 5.

[*]Department of Mathematics, Yokohama National University, 79-2 Tokiwadai, Hodogaya-ku, Yokohama 240-8501, Japan Email: `nakamoto@edhs.ynu.ac.jp`

[†]Department of Information Media and Environment Sciences, Graduate School of Environment and Information Sciences, Yokohama National University, 79-7 Tokiwadai, Hodogaya-ku, Yokohama 240-8501, Japan

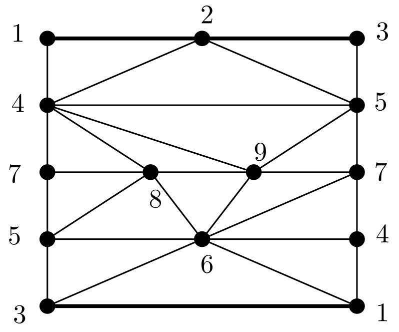


Figure 1: A Möbius triangulation with no geometric realization constructed by Brehm

Let us consider nonorientable surfaces, in particular, the projective plane. Since the projective plane itself is not embeddable in $\mathbb{R}^3$, no map on the projective plane has a geometric realization. However, the surface obtained from the projective plane by removing an open disk (i.e., a *Möbius band*) is embeddable in $\mathbb{R}^3$, and hence we can expect that a triangulation on the Möbius band has a geometric realization.

For simple notations, we call a triangulation on the projective plane and that on the Möbius band a *projective triangulation* and a *Möbius triangulation*, respectively, throughout the paper.

In the current work, we discuss geometric realizations of Möbius triangulations. However, Brehm [4] has already found a Möbius triangulation with no geometric realization, which is shown in Figure 1. (In Figure 1, identify vertices with the same label.) Can we get an affirmative result for geometric realizations of Möbius triangulations?

Let G be a projective triangulation and let f be a face of G. Let $G - f$ denote the Möbius triangulation obtained from G by removing the interior of f. Recently, the following has been proved:

Theorem 1 (Bonnington and Nakamoto [2]) *Every projective triangulation G has a face f such that $G - f$ has a geometric realization.*

For example, the smallest projective triangulation has exactly six vertices and the graph is isomorphic to a complete graph K_6 with six vertices. Figure 2 shows it in the left-hand side, and in the right-hand side, the triangulation minus face 256 has a geometric realization.

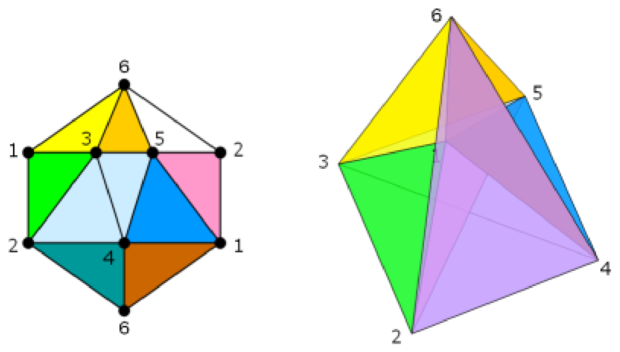


Figure 2: A geometric realization of $K_6 - \triangle 256$

As far as we know, Theorem 1 seems to be the first affirmative result for geometric realizations of maps on nonorientable surfaces since Brehm found the counterexample shown in Figure 1.

Which face f of a projective triangulation G makes $G-f$ geometrically realizable? Brehm's counterexample implicitly assures that a projective triangulation G has no geometric realization if G has a face f and a 3-cycle C such that the boundary cycle of f and C are disjoint and bound an annular region. In this case, we say that C is a *nesting* 3-cycle of f. (In Figure 1, if 123 is the boundary of f, then 456 is a nesting cycle C of f.)

In the current work, we shall characterize a face f in a projective triangulation G such that $G - f$ has a geometric realization, as follows.

Theorem 2 *Let G be a projective triangulation and let f be a face of G. Then $G - f$ has a geometric realization if and only if there is no nesting 3-cycle of f in G.*

Theorem 2 asserts that only the structure of Brehm's example with a boundary 3-cycle and its nesting 3-cycle breaks the geometric realizability of Möbius triangulations. Hence we have the following corollary which answers a question by Archdeacon et al. in [1].

Corollary 3 *A Möbius triangulation M has a geometric realization unless M has two disjoint 3-cycles homotopic to the boundary of M.*

Moreover, Theorem 2 implies Theorem 1, since we can easily prove that any projective triangulation has a face with no nesting 3-cycle. Furthermore, the following corollary immediately follows, since a 4-connected projective triangulation has no face with a nesting 3-cycle.

Corollary 4 *Let G be a 4-connected projective triangulation. Then, for any face f of G, $G - f$ has a geometric realization.*

A weaker version of Corollary 4 has already been known as in the following, but Corollary 4 improved it with respect to the connectivity of triangulations.

Theorem 5 (Chávez et al. [5]) *Let G be a 5-connected projective triangulation. The, for any face f of G, $G - f$ has a geometric realization.*

Corollary 4 can be strengthened by introducing a notion of the cyclical k-connectivity. A graph H is said to be *cyclically k-connected* if H has no k-cut S such that each component of $H - S$ has a cycle. Then we can improve Theorem 5 with a necessary and sufficient condition, as follows. This solved a conjecture given in [2, 5]

Corollary 6 *Let G be a projective triangulation and let f be a face of G. Then, for any face f of G, $G - f$ has a geometric realization if and only if G is cyclically 4-connected.*

2 Procedures in the proof

In this section, we describe our procedures to prove Theorem 2. Let G be a projective triangulation, and let f be any face of G with no nesting 3-cycle of f. We shall construct a geometric realization of $G - f$, as follows.

PROCEDURE 1. Choose any face, say f', neighboring to f, and let G' be the map on the Möbius band obtained from G by removing the 2-cell region $f \cup f'$.

Let G be a map on a surface and let xy be an edge of G. *Contraction* of xy is to remove xy and identify x and y. (If the resulting map has a face bounded by a 2-cycle, then we replace the multiple edge with a single edge.) Its inverse operation is called a *splitting* of a vertex $[xy]$, where $[xy]$ is the image of the edge xy by the contraction.

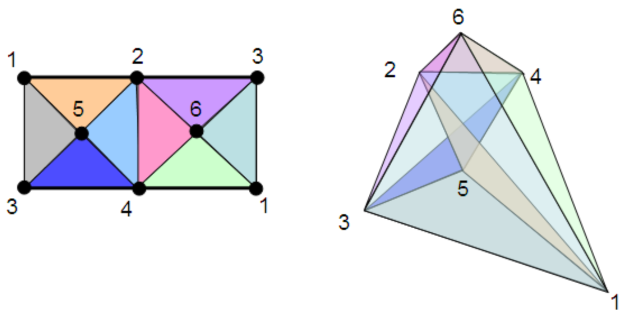


Figure 3: The map K on the Möbius band

Let K be the map on a Möbius band whose graph is isomorphic to K_6 minus two independent edges and whose boundary has a 4-cycle, as shown in Figure 3. A *split-K* is a map obtained from K by splitting some of the vertices of K, and subdividing edges by vertices of degree 2, but its boundary must have length exactly four. We call each vertex of K' with degree at least 3 a *node* of K'. A *segment* of K' means a path of K' joining two nodes and containing no node.

PROCEDURE 2. Prove that G' can be transformed into K by contracting and deleting edges not on the boundary. Then we can get a split-K, denoted by K', whose boundary coincides with that of G'. Moreover, each segment of K' can be taken to be *induced*, i.e., without a chord.

Note that if f has a nesting 3-cycle, then Procedure 2 is impossible by the obstruction of the 3-cycle.

Let us introduce a notion called "an exhibition of a map", which will play an important role to construct a geometric realization of a given map.

Let R be a *near triangulation*, that is, a 2-connected plane graph with each finite face triangular. Let C be the *boundary* of R, that is, the cycle of R bounding the infinite face. Suppose that we are given an embedding of C in $\mathbb{R}^3$ with each edge a straight-line segment, and that there exists a plane $P \subset \mathbb{R}^3$ such that the image of C in $\mathbb{R}^3$ is projected to the plane P as a convex polygon. Such an embedding of C in $\mathbb{R}^3$ is called an *exhibition*. The following is an important lemma for constructing a geometric realization of a map.

Lemma 7 ([1, 2]) *Let R be a near triangulation and let C be the boundary of R. Suppose that an exhibition of C in $\mathbb{R}^3$ is given. Then the exhibition of C extends to a geometric realization in the convex hull of C in $\mathbb{R}^3$.*

Let us construct a geometric realization of $G - f$. In particular, we shall construct a geometric realization of G' for which we can add the face f' without introducing any internal collision of faces. So, by Lemma 7, it suffices to guarantee the following procedure is possible, since each face of K' corresponds to a near triangulation contained in G.

Let M be a triangulation on a surface F^2 and let H be a subgraph of M each of whose segment is induced, where a *segment* of H means a path of M joining two vertices of H of degree at least 3 and containing no inner vertices of H of degree at least 3.) An *exhibition* of H is an embedding of F^2 in $\mathbb{R}^3$ such that

(i) each segment of H is a straight segment in $\mathbb{R}^3$,

(ii) the boundary of each face of H gives an exhibition in $\mathbb{R}^3$,

(iii) for any distinct two faces of H, the convex hulls of their exhibitions are internally disjoint in $\mathbb{R}^3$.

By Lemma 7 and the definition of an exhibition of a map, in order to construct a geometric realization of a triangulation M, it suffices to find a subgraph H of M which has an exhibition in $\mathbb{R}^3$.

PROCEDURE 3. Let K' be a split-K and let $v_1v_2v_3v_4$ be the boundary 4-cycle of K'. Prove that K' has an exhibition, and that we can add a triangular disk $v_1v_2v_3$ to the geometric realization of G' without an internal collision of faces.

3 Construction of geometric realization

The procedures to construct a geometric realization of a projective triangulation minus a face have been mentioned in the previous section. In this section, we describe what is a difficulty in our proof.

Roughly speaking, in the proof of Theorems 1 and 5, a K_6*-minor* (i.e., a map transformed into K_6 by contracting and deleting edges) plays the same role as the map K, and a geometric realization of a K_6-minor can be easily constructed as in Figure 2. In these two theorems, since the conclusion is to be able to construct a geometric realization of a map after removing *some* face of G, or since we can assume the 5-connectedness of graphs, we have been able to detect a K_6-minor in G which is a relatively *dense* map on the projective plane. Such a graph can easily be geometrically realized as we expect.

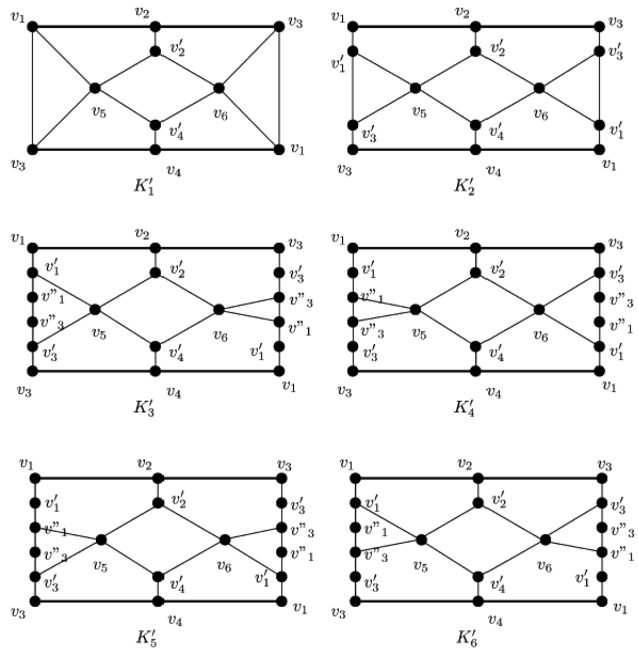


Figure 4: Split-K' on the Möbius band

On the other hand, in order to prove Theorem 2, we can only assume the non-existence of a nesting 3-cycle of a face f arbitrarily chosen in G. Hence we can only detect a *sparse* structure of G, which is a split-K in Procedure 2. Since a split-K is so sparse, we have to consider many cases for constructing an exhibition of the split-K in Procedure 3.

In Figure 4, we list all possible homeomorphism types of split-K's. Each of them must be dealt with independently, since a minor relation of maps is not compatible with geometric realizability, that is, even if a map P is geometrically realizable, a geometric realization of a map Q with $Q \geq_m P$ cannot be obtained from that of P easily.

Let us put an example of a geometric realization of K_1'. In order to do so, we have only to construct an exhibition of K_1', as described in Procedure 3. That is, we give an exhibition of each face of K_1' such that for any distinct faces, the convex hull corresponding to them does not cross internally. In particular, giving the following 3-dimensional $\mathbb{R}^3$-coordinates to each vertex, we can construct an exhibition: For K_1',

$$\begin{array}{ll} v_1 = (12, -10, 5), & v_2 = (-10, 10, -3), \\ v_2' = (-9, 10, -2), & v_3 = (-20, -10, 4), \\ v_4 = (4, 10, -5), & v_4' = (5, 10, -6), \\ v_5 = (0, 0, 0), & v_6 = (0, 40, 0). \end{array}$$

Then we obtain the exhibition shown in Figure 5.

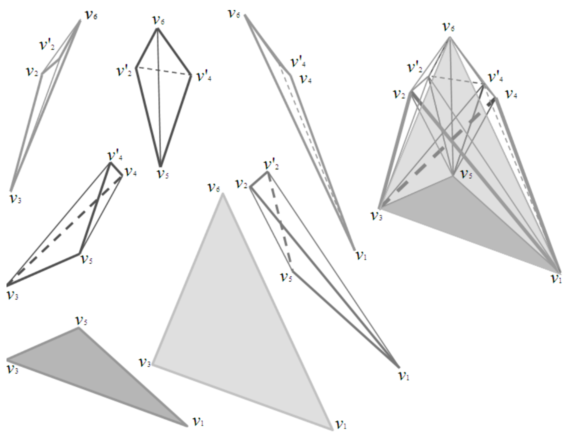


Figure 5: The map K_1' on the Möbius band

For K_i' with $i = 2, 3, 4, 5, 6$, we have to construct exhibitions of them independently. As the reader see, it is too complicated to grasp the complete structure only by figures, that is, we cannot check whether this really gives an exhibition. So, in the paper, using a computer program, we have checked that our coordinates for K_i' really gives an exhibition of K_i', for $i = 1, 2, 3, 4, 5, 6$.

4 Conclusion

Historically there are many papers dealing with geometric realization of orientable maps, but not with nonorientable maps. We do not know the reason why, but it might be by the Brehm's example, or because a nonorientable closed surface is not embeddable in $\mathbb{R}^3$ and the problem for nonorientable surface is not so natural as the orientable case.

Theorem 1 asserts that in a given projective triangulation G, if we choose a face f carefully, then $G - f$ has a geometric realization. However, this theorem says nothing about a characterization of geometrically realizable Möbius triangulations.

In order to answer this, Chávez et al. [5] got a result that a Möbius triangulation is geometrically realizable if it is a subgraph of a 5-connected projective triangulation (Theorem 5). In proving the result, they used a skillful argument for finding a K_6-minor with a good property in a projective triangulation G with any face f specified. However, this did not reach a complete characterization of a face f with $G - f$ geometrically realizable.

The current result (Theorem 2) solves the problems on geometric realization of a projective triangulation with one face removed and that of a Möbius triangulation which have been considered in the two papers [2, 5].

References

[1] D. Archdeacon, C.P. Bonnington and J.A. Ellis-Monanghan, How to exhibit toroidal maps in space, *Discrete Comp. Geom.* **38** (2007), 573–594.

[2] P. Bonnington and A. Nakamoto, Geometric realization of a projective triangulation with one face removed, *Discrete Comp. Geom.* **40** (2008), 141–157.

[3] J. Bokowski and A. Guedes de Oliveira, On the generation of oriented matroids, *Discrete Comput. Geom.* **24** (2004), 197–208.

[4] U. Brehm, A nonpolyhedral triangulated Möbius strip, *Proc. Amer. Math. Soc.* **89** (1983), 519–522.

[5] M. Chávez, G. Fijavž, A. Márquez, A. Nakamoto and Suárez, Geometric realization of triangulations on the Möbius band, *SIAM J. Discrete Math.* **23** (2008), 221–232.

[6] B. Grünbaum, "Convex polytopes", *Pure and Applied Mathematics* Vol. 16, Interscience-Wiley, New York, 1967.

[7] E. Steinitz, Polyeder und Raumeinteilungen, *Enzykl. Math. Wiss.* Vol. 3, Teil 3A612 (1922), 1–139.

Multi-pseudotriangulations

Vincent Pilaud* Michel Pocchiola*

Abstract

We introduce a natural generalization of both pseudotriangulations and multitriangulations, that we call *multi-pseudotriangulations*. We propose an enumeration algorithm for multi-pseudotriangulations, based on certain greedy multi-pseudotriangulations that are closely related with sorting networks.

The proofs of the results of this extended abstract are skipped for space reasons.

1 Definition

Let $\mathcal{M}$ denote the *Möbius strip* (without boundary), that is, the quotient set of $\mathbb{R}^2$ by the relation $(x, y) \sim (x + \pi, -y)$. Let π denote the canonical projection.

A *pseudoline* is a non-separating simple closed curve in $\mathcal{M}$. Throughout this paper, a pseudoline is always assumed to be *monotone*, and thus can be seen as the image under the projection π of the graph $\{(x, f(x)) \,|\, x \in \mathbb{R}\}$ of a continuous and π-antiperiodic function $f : \mathbb{R} \to \mathbb{R}$.

A *pseudoline arrangement* (see Fig. 1) is a finite set of pseudolines such that any two of them have exactly one crossing point and possibly some contact points. We are only interested in *simple* arrangements, that is, where no three pseudolines meet in a common point. The *support* of a pseudoline arrangement is the union of its pseudolines. The *first level* of a pseudoline arrangement is its external hull, that is, the boundary of the cell at infinity defined by the support of the arrangement. Similarly, we define recursively the *pth level* of an arrangement to be the external hull of (the closure of) its support minus its first $p - 1$ levels.

The main objects with which this paper will be dealing are the following (see Fig. 1):

Definition 1 *Let L be an arrangement of n pseudolines and k be an integer. A* k-pseudotriangulation *of L is a set U of vertices of L containing all contact points of L and obtained as the union of*

1. *the set of vertices of the first k levels of L, and*
2. *the set of contact points of an arrangement U' of $n - 2k$ pseudolines whose support covers the support of L minus its first k levels.*

In this definition, observe that U' is completely determined by U (and *vice versa*). We denote it $\Lambda(U)$.

*Département d'Informatique, École Normale Supérieure, Paris, France, {vincent.pilaud,michel.pocchiola}@ens.fr

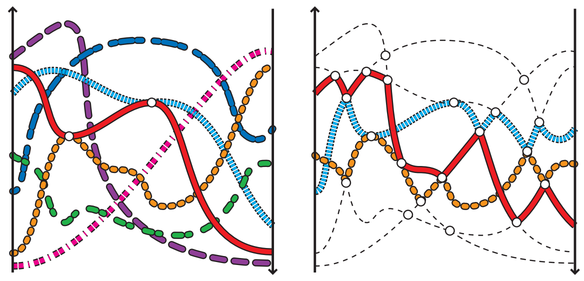


Figure 1: An arrangement of seven pseudolines and a 2-pseudotriangulation of it.

2 Pseudotriangulations and multitriangulations

Duality Remember that the Möbius strip $\mathcal{M}$ is the line space of the plane : we parametrize a line by its angle with the horizontal axis and its algebraic distance to the origin. Via this parametrization, the set of lines passing through a point p of the plane forms a pseudoline p^* of $\mathcal{M}$. If P is a point set in general position, then the set $P^* = \{p^* \,|\, p \in P\}$ is a simple pseudoline arrangement (without contact points).

In this section, we use the following notation: if p and q are two points of P, e is the edge pq and u is the crossing point of p^* and q^* (*i.e.*, the line pq), then we denote $e^{\succ} = u$ and $u^{\prec} = e$. We also denote $E^{\succ} = \{e^{\succ} \,|\, e \in E\}$ and $U^{\prec} = \{u^{\prec} \,|\, u \in U\}$.

Pseudotriangulations Let P be a point set in general position. A set E of edges of $\binom{P}{2}$ is *pointed* if for any vertex $p \in P$, there exists a line which passes through p and defines a half-plane containing all the edges of E incident to p. A *pseudotriangulation* of P is a maximal crossing-free pointed set of edges of $\binom{P}{2}$ [8, 9, 10, 11]. The following theorem is one of our main motivations, and justifies the title of this paper:

Theorem 1 1. *If T is a pseudotriangulation of P, then $T^{\succ}$ is a 1-pseudotriangulation of P^*.*

2. *If U is a 1-pseudotriangulation of P^*, then $U^{\prec}$ is a pseudotriangulation of P.*

To make clear the previous theorem, let us interpret the pseudolines of $\Lambda(T^{\succ})$. Remember that the pseudotriangulation T decomposes the convex hull of P into *pseudotriangles* (plane polygons with only three convex vertices, joined by three concave polygonal chains; see Fig. 2). A pseudoline of $\Lambda(T^{\succ})$ is the set of all inner tangents to a pseudotriangle of T.

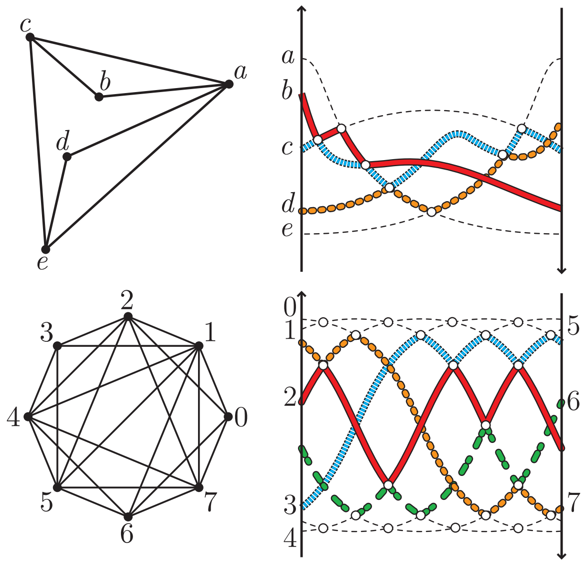


Figure 2: A pseudotriangulation and its dual arrangement. A 2-triangulation and its dual arrangement.

Multitriangulations Let P be a point set in convex position. For $\ell \in \mathbb{N}$, an *ℓ-crossing* is a set of ℓ mutually crossing edges. A *k-triangulation* of P is a maximal $(k+1)$-crossing-free subset of edges of $\binom{P}{2}$ [6, 3, 7]. Again, there is a duality between multi-pseudotriangulations of P and multitriangulations:

Theorem 2 *1. If T is a k-triangulation of P, then $T^{\succ}$ is a k-pseudotriangulation of P^*.*

2. If U is a k-pseudotriangulation of P^*, then $U^{\prec}$ is a k-triangulation of P.

The pseudolines of $\Lambda(T^{\succ})$ are easy to interpret with the following natural generalization of triangles introduced in [7]: a k-star is a set of edges of the form $\{s_j s_{j+k} \mid j \in \mathbb{Z}_{2k+1}\}$, where $s_0, \dots, s_{2k}$ are cyclically ordered. A pseudoline of $\Lambda(T^{\succ})$ is the set of all bisectors of a k-star of T.

3 Flips and greedy multi-pseudotriangulations

Flips Let L be a pseudoline arrangement without contact points. Let U be a k-pseudotriangulation of L. Let $u \in U$ be the contact point of two pseudolines of $\Lambda(U)$, and v denote their crossing point. Then that $U \triangle \{u, v\}$ is a new k-pseudotriangulation of L (we use the symbol $\triangle$ for the symmetric difference: $U \triangle \{u, v\} = U \cup \{v\} \smallsetminus \{u\}$). We say that we obtain $U \triangle \{u, v\}$ from U by flipping u. The primal notions of flips (for pseudotriangulations and for multitriangulations) obviously correspond to this dual version.

Let $G^k(L)$ be the graph whose vertices are the k-pseudotriangulations of L and whose edges are the pairs of k-pseudotriangulations linked by a flip.

Theorem 3 *The graph $G^k(L)$ is connected.*

Cuts A *cut* of L is a pseudoline λ of $\bar{\mathcal{M}} = \mathcal{M} \cup \{\infty\}$ passing through the point at infinity and such that $L \cup \{\lambda\}$ is a simple pseudoline arrangement of $\bar{\mathcal{M}}$. We orient all the pseudolines of L in the same arbitrary direction (*e.g.*, the abscisse increasing direction). Then, a cut λ naturally defines a partial order $\preccurlyeq_\lambda$ on the set of vertices of L: $u \preccurlyeq_\lambda v$ if there exists an oriented path on the support of L from u to v and avoiding λ.

Let U be a k-pseudotriangulation of L, u be a flippable contact point of U, and v denote the corresponding crossing point. It is easy to see that u and v are comparable for $\preccurlyeq_\lambda$. We say that the flip of u is *λ-increasing* if $u \preccurlyeq_\lambda v$ and *λ-decreasing* otherwise. We denote $G^k_\lambda(L)$ the directed graph of λ-increasing flips on k-pseudotriangulations of L. This directed graph is acyclic. In the following paragraph, we characterize its unique sink in terms of sorting networks.

Sorting networks We cut the Möbius strip along the cut λ, and denote λ_- and λ_+ the two copies of λ such that an oriented path on the support of L goes from λ_- to λ_+. We orient λ_- and λ_+ in the same arbitrary direction.

A *sweep* of L is a pencil $\lambda_0, \dots, \lambda_N$ of cuts of L such that for all $i \geq 1$, λ_i is obtained from λ_{i-1} by sweeping a crossing point u_i of L, minimal for the order $\preccurlyeq_{\lambda_{i-1}}$. We consider a sweep $\lambda_0, \dots, \lambda_N$ of L with $\lambda_0 = \lambda_-$ and $\lambda_N = \lambda_+$ (thus, $N = \binom{|L|}{2}$). We orient all pseudolines $\lambda_1, \dots, \lambda_{N-1}$ in the same direction as λ_- and λ_+. We denote p_i the integer such that the pseudolines of L that cross at u_i are the p_ith and (p_i+1)th pseudolines of L on λ_i.

Let U be a k-pseudotriangulation of L. Since we have cut $\mathcal{M}$ along λ, each of the first k levels of L is divided into two pieces. Thus, the support of L is decomposed into $n = |L|$ curves: the $2k$ pieces of the first k levels of L and the $n - 2k$ pseudolines of $\Lambda(U)$. We denote these curves $\ell_1, \dots, \ell_n$ in the order they reach λ_+. With this convention, the order in which the curves leave λ_- is given by the permutation τ_k of $\{1, \dots, n\}$ defined by $\tau_k(i) = n - i$ if $i \in \{k+1, \dots, n-k\}$, and $\tau_k(i) = i$ otherwise (see Fig. 3).

To any oriented cut μ of L, we associate the permutation $\pi_{\mu,U}$ of $\{1, \dots, n\}$ given by the order of $\ell_1, \dots, \ell_n$ on μ. For example, $\pi_{\lambda_+,U}$ is the identity permutation, while $\pi_{\lambda_-,U}$ is the permutation τ_k. We consider the sequence of permutations $(\pi_{\lambda_i,U})_{0 \leq i \leq N}$. By definition, this sequence starts with τ_k, and ends with the identity permutation. Furthermore, for all i,

1. if u_i is in U, then $\pi_{\lambda_i,U} = \pi_{\lambda_{i-1},U}$
2. otherwise, $\pi_{\lambda_i,U}$ is obtained from $\pi_{\lambda_{i-1},U}$ by inverting its p_ith and (p_i+1)th entries.

Theorem 4 *The directed graph of flips $G^k_\lambda(L)$ has a unique sink U characterized by the property that for all i, the permutation $\pi_{\lambda_i,U}$ is obtained from $\pi_{\lambda_{i-1},U}$ by sorting its p_ith and (p_i+1)th entries.*

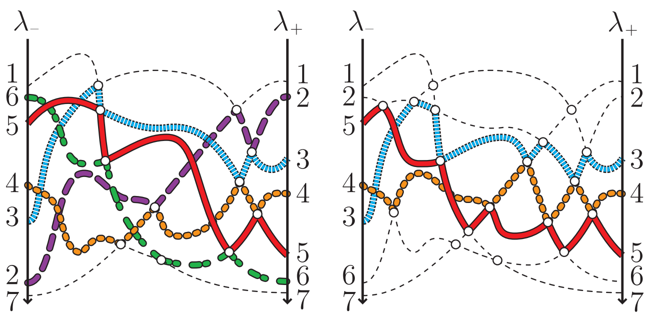


Figure 3: Greedy multi-pseudotriangulations of the pseudoline arrangement of Fig. 1.

Let us reformulate Theorem 4 in the context of sorting networks [4, Section 5.3.4]. Let $i < j$ be two integers. A *comparator* $[i : j]$ transforms a sequence of numbers $(x_1, \dots, x_n)$ by sorting (x_i, x_j), *i.e.*, replacing x_i by $\min(x_i, x_j)$ and x_j by $\max(x_i, x_j)$. A comparator $[i : j]$ is *primitive* if $j = i + 1$. A *sorting network* is a sequence of comparators that sorts any sequence $(x_1, \dots, x_n)$.

A pseudoline arrangement L together with a sweep $\lambda_0, \dots, \lambda_N$ define a sequence of primitive comparators $S_L = [p_1 : p_1 + 1], \dots, [p_N : p_N + 1]$ [5, Section 8]. In this setting, Theorem 4 affirms that,

1. given a pseudoline arrangement L, a cut λ of L and any sweep starting at λ_- and ending at λ_+, sorting the permutation τ_k according to S_L provides a k-pseudotriangulation of L.
2. this k-pseudotriangulation only depends upon L, k, and λ (not on the total order given by the sweep).

We call this k-pseudotriangulation the *λ-greedy k-pseudotriangulation of* L and denote it $\Gamma^k_\lambda(L)$.

When $k = 1$, the greedy 1-pseudotriangulation is in fact the dual of the *greedy pseudotriangulation* of [1] (originally [8] for convex bodies). This pseudotriangulation is obtained by a recursive choice of edges: we start from the empty set and add at each step a maximal (for $\preccurlyeq_\lambda$) remaining edge e_i such that the set of edges $\{e_1, \dots, e_i\}$ remains pointed and non-crossing, until we obtain a pseudotriangulation.

Constrained greedy multi-pseudotriangulation Let V be a subset of crossing points of L. We denote by $G^k_\lambda(L \,|\, V)$ the subgraph of $G^k_\lambda(L)$ induced by the k-pseudotriangulations of L that contain V.

Theorem 5 *The directed graph of flips $G^k_\lambda(L \,|\, V)$ either is empty or has a unique sink U characterized by the property that for all i,*

1. *if $u_i \in V$, then $\pi_{\lambda_i,U} = \pi_{\lambda_{i-1},U}$*
2. *if $u_i \notin V$, then $\pi_{\lambda_i,U}$ is obtained from $\pi_{\lambda_{i-1},U}$ by sorting its p_ith and $(p_i + 1)$th entries.*

We denote $\Gamma^k_\lambda(L \,|\, V)$ this unique sink.

4 Greedy flip property and enumeration

We are now ready to state the *greedy flip property*, that says how to update the greedy k-pseudotriangulation of L when the cut λ varies:

Theorem 6 (Greedy Flip Property) *Let L be a pseudoline arrangement. Let λ be a cut of L, and μ be a cut of L obtained from λ by sweeping a maximal (for $\preccurlyeq_\lambda$) vertex v of L. Let V be a set of crossing points of L contained in at least one k-pseudotriangulation of L. Then,*

1. *if v is in $\Gamma^k_\lambda(L \,|\, V)$, but neither in V, nor in the first k levels of L, then $\Gamma^k_\mu(L \,|\, V)$ is obtained from $\Gamma^k_\lambda(L \,|\, V)$ by flipping v. Otherwise, $\Gamma^k_\mu(L \,|\, V) = \Gamma^k_\lambda(L \,|\, V)$;*
2. *if v is in $\Gamma^k_\lambda(L \,|\, V)$, then $\Gamma^k_\mu(L \,|\, V \cup \{v\}) = \Gamma^k_\lambda(L \,|\, V)$. Otherwise, $V \cup \{v\}$ is not contained in a k-pseudotriangulation of L.*

From this greedy flip property, we derive an algorithm to enumerate multi-pseudotriangulations, similar to the enumeration algorithm of [1].

Let L be a pseudoline arrangement and λ be a cut of L. We say that a k-pseudotriangulation of L is *colored* if its contact points are colored either in blue or in red. A red contact point is fixed for the end of the algorithm, while a blue one can be flipped. In the algorithm, we construct a binary tree $\mathcal{T}$, whose nodes are colored k-pseudotriangulations of L, as follows:

1. the root of the tree is the λ-greedy k-pseudotriangulation of L, entirely colored in blue;
2. for any colored k-pseudotriangulation U of L, if there is no λ-decreasing flippable point (*i.e.*, if there only remains red contact points, or if the flip of any blue contact point of U is λ-increasing), then U is a leaf of $\mathcal{T}$;
3. otherwise, we choose a maximal blue point u of U whose flip is λ-decreasing. The right child of U is obtained by flipping u and its left child is obtained by changing the color of u into red.

This algorithm depends upon what maximal blue point we choose at each step. However, no matter what this choice is, we obtain all k-pseudotriangulations of L:

Theorem 7 *The set of k-pseudotriangulations of L is exactly the set of red-colored leafs of $\mathcal{T}$.*

Let us briefly discuss the complexity of this algorithm. We assume that the input of the algorithm is a pseudoline arrangement and we consider a flip as an elementary operation. Then, this algorithm requires a polynomial running time per k-pseudotriangulation. As for many enumeration algorithms, the crucial point of this algorithm is that its working space is also polynomial (while the number of k-pseudotriangulations of L is exponential for fixed k).

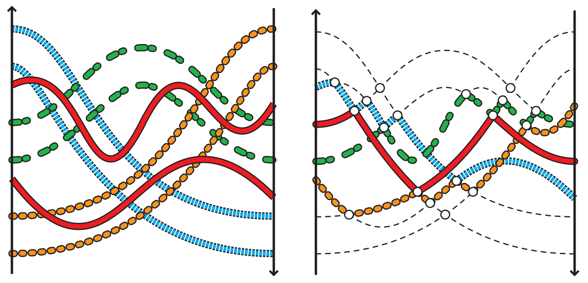


Figure 4: A double pseudoline arrangement and a 2-pseudotriangulation of it.

5 Further topics and open questions

Double pseudoline arrangements A double pseudoline is a non-contractible and non-separating simple closed curve in $\mathcal{M}$ (see Fig. 4). A *double pseudoline arrangement* [2] is a finite set of double pseudolines such that any two of them

1. have exactly four intersection points and cross transversally at these points; and
2. induce a cell decomposition of the Möbius strip.

The set C^* of tangents to a convex body C of the plane is a double pseudoline. If Q is a set of pairwise disjoint convex bodies, then $Q^* = \{C^* \mid C \in Q\}$ is a double pseudoline arrangement.

Defining similarly the *support* and the *levels* of a double pseudoline arrangement, we can define multi-pseudotriangulations of double pseudoline arrangements (see Fig. 4): a *k-pseudotriangulation* of an arrangement L of n double pseudolines is the set U of contact points (including the vertices of the first k levels of L) of an arrangement $\Lambda(U)$ of $2n-2k$ pseudolines such that the support of $\Lambda(U)$ covers the support of L minus its first k levels (see Fig. 4d).

Multi-pseudotriangulations of double pseudoline arrangements have similar structural properties and can also be enumerated by the greedy flip algorithm.

The graph of flips Let L be a pseudoline arrangement and k be an integer. Let $\Theta^k(L)$ denote the partially ordered set of sets of contact points of L that contain the vertices of the first k levels and the contact points of L and that are contained in a k-pseudotriangulation of L, ordered by reverse inclusion. Theorems 3 and 5 ensure that $\Theta^k(L)$ is an abstract polytope whose 1-skeleton is the graph of flips $G^k(L)$ (see the discussion in [1, Subsection 2.2]). When $k=1$ and L is the dual pseudoline arrangement of a point set of the Euclidean plane, it turns out that this abstract polytope can be realized effectively as a polytope of $\mathbb{R}^d$ (where d is the number of flippable edges), known as the *polytope of pointed pseudotriangulations* [9]. We naturally would like to know whether $\Theta^k(L)$ is a polytope in general.

Acknowledgments

We thank Francisco Santos and Micha Sharir for fruitful discussions on the subject.

References

[1] H. Brönnimann, L. Kettner, M. Pocchiola, and J. Snoeyink. Counting and enumerating pointed pseudotriangulations with the greedy flip algorithm. *SIAM J. Comput.*, 36(3):721–739, 2006. Preliminary version in *Proc. 7th Workshop on Algorithm Engineering and Experiments*, pages 98–110, 2005.

[2] L. Habert and M. Pocchiola. Arrangements of double pseudolines in Möbius strips. Submitted to *Discrete Comput. Geom.* Abbreviated version in *Abstracts 12th European Workshop Comut. Geom.*, pages 211–214, 2006. Poster version presented at the *Workshop on Geometric and Topological Combinatorics* (satellite conference of ICM 2006).

[3] J. Jonsson. Generalized triangulations and diagonal-free subsets of stack polominoes. *J. Combin. Theory Ser. A*, 112(1):117–142, 2005.

[4] D. E. Knuth. *The art of computer programming. Volume 3.* Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1973. Sorting and searching, Addison-Wesley Series in Computer Science and Information Processing.

[5] D. E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, Germany, 1992.

[6] T. Nakamigawa. A generalization of diagonal flips in a convex polygon. *Theoret. Comput. Sci.*, 235(2):271–282, 2000.

[7] V. Pilaud and F. Santos. Multi-triangulations as complexes of star-polygons. to appear in *Discrete Comput. Geom.*, 2008.

[8] M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudotriangulations. *Discrete Comput. Geom.*, 16(4):419–453, 1996. Special issue for *Proc. 11th Annu. ACM Sympos. Comput. Geom.*

[9] G. Rote, F. Santos, and I. Streinu. Expansive motions and the polytope of pointed pseudo-triangulations. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*, volume 25 of *Algorithms Combin.*, pages 699–736. Springer, 2003.

[10] G. Rote, F. Santos, and I. Streinu. Pseudo-triangulations - a survey. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, volume 453 of *Contemporary Mathematics*, pages 343–410. Amer. Math. Soc., 2008.

[11] I. Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete Comput. Geom.*, 34:587–635, 2005. Preliminary version in *Proc. 41st Annu. IEEE Symp. Found. Comput. Sci.*, pages 443–453, 2000.

Fast Delaunay Triangulation for Converging Point Relocation Sequences

Pedro Machado Manhães de Castro* Olivier Devillers*

Abstract

This paper considers the problem of updating efficiently a Delaunay triangulation when vertices are moving under small perturbations. Its main contribution is a set of algorithms based on the concept of vertex tolerance. Experiments show that it is able to outperform the naive *rebuilding* algorithm in certain conditions. For instance, when points, in two dimensions, are relocated by Lloyd's iterations, our algorithm performs several times faster than *rebuilding*.

1 Introduction

Delaunay triangulation of a set of points is one of the most famous data structures produced by computational geometry. Two main reasons explain this success: –1– computational geometers eventually produce efficient algorithms to compute it, and –2– it has many practical uses such as meshing for finite elements methods or surface reconstruction from point clouds.

For several applications the data are moving and thus the triangulation evolves with time. It arises for example when meshing deformable objects [4], or in some algorithms relocating the points by variational methods [1].

We first recall that the *Delaunay triangulation* $DT(S)$ of a set S of n points in $\mathbb{R}^d$ is a *simplicial complex* such that no point in S is inside the circumsphere of any *simplex* in $DT(S)$. Several algorithms to compute the Delaunay triangulation are available in the literature. Many of them work in the *static* setting, where the points are fixed and known in advance. There are also a variety of so-called *dynamic* algorithms [5], in which the points are fixed but not known in advance and thus the triangulation is maintained under point insertions or deletions. If some of the points move continuously in $\mathbb{R}^d$ and we want to keep track of the modifications of the triangulation, we are dealing with *kinetic* algorithms [8]. Finally, an important variation is when the points move but we are only interested in the triangulation at some discrete times, we call that context *timestamps relocation*.

When we are in the context of timestamps relocation, a simple and efficient method to consider is the following: for each timestamp we simply recompute the Delaunay triangulation of the set $S(t)$ at timestamp t_i. We call this algorithm *rebuilding*. Note that this algorithm does not take any previous work into account, thus for timestamp t_i it does not benefit from any possible correlation between $S(t_j)$ and $S(t_i)$, with $t_j < t_i$. If points are well-distributed, it could achieve an $O(kn\log(n))$ computation time, where n is the number of vertices on the triangulation and k is the number of distinct timestamps.

Although the rebuilding algorithm is naive and has a poor theoretical complexity, it stands for an algorithm hard to outperform when most of the points move, as observed in previous work [8]. In this paper, we propose to compute for each point a safety zone where the point can move without changing its connectivity in the triangulation. Several experiments conducted on synthetic and practical data show added value of the method, in particular for mesh smoothing [2] where the points are converging to a final position.

2 Safe regions

Given any certificate $C : \mathcal{A}^m \rightarrow \{-1, 0, 1\}$ acting on a m-tuple of points $\zeta = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m) \in \mathcal{A}^m$, where $\mathcal{A}$ is the space where the input lies in, we define the *tolerance* of C with respect to ζ, namely $\epsilon_C(\zeta)$ or simply $\epsilon(\zeta)$ when there is no ambiguity, the largest displacement applicable to $\mathbf{z}$ in ζ without invalidating C. Hereafter, a certificate is *valid* when it is positive. Then, more precisely, the tolerance can be stated as follows:

$$\epsilon_C(\zeta) = \inf_{\substack{\zeta' \\ C(\zeta') \leq 0}} dist_H(\zeta, \zeta'), \tag{1}$$

where $dist_H(\zeta, \zeta')$ is the Hausdorff distance between two finite sets of points.

By abuse of notation, $\mathbf{z} \in \zeta$ means that $\mathbf{z}$ is one of the points of ζ. Let $\mathcal{X}$ be a finite set of m-tuples of points in $\mathcal{A}^m$, then the *tolerance of an element* $\mathbf{e} \in \mathcal{A}$ with respect to a given certificate C and $\mathcal{X}$, namely $\epsilon_{C,\mathcal{X}}(\mathbf{e})$ or simply $\epsilon(\mathbf{e})$ when there is no ambiguity, can be defined as follows:

$$\epsilon_{C,\mathcal{X}}(\mathbf{e}) = \inf_{\substack{\zeta \ni \mathbf{e} \\ \zeta \in \mathcal{X}}} \epsilon_C(\zeta). \tag{2}$$

The tolerance involved in a Delaunay triangulation is the *tolerance of the empty-sphere certificate* acting on any bi-cell of a Delaunay triangulation. From Equation 1, it corresponds to the size of the smallest perturbation the bi-cell's vertices could undergo so as to become cospherical.

*Email: First.Lastname@sophia.inria.fr. INRIA, BP93, 06902 Sophia-Antipolis, France, www.inria.fr/sophia/members/First.Lastname. The authors wish to thank the ANR Triangles, contract number ANR-07-BLAN-0319, and region PACA for their support.

This is equivalent to compute the hypersphere that minimizes the maximum distance to the $d+2$ vertices, or to compute half the width of the d*-annulus of minimum width* containing the vertices.

Let $\mathcal{T}$ be a triangulation lying in $\mathbb{R}^d$ and $\mathcal{B}$ a bi-cell in $\mathcal{T}$. The *interior facet* of $\mathcal{B}$ is the common facet of the two cells of $\mathcal{B}$. The *opposing vertices* of $\mathcal{B}$ are the remaining two vertices that do not belong to its interior facet. Two bi-cells $\mathcal{B}, \mathcal{B}'$ are *neighbors* if they share a cell. If the interior facet and opposing vertices of $\mathcal{B}$ are respectively inside and outside (or on) a common hypersphere $\mathcal{S}$, we say that $\mathcal{B}$ verifies the *safety condition*. We call $\mathcal{B}$ a *safe bi-cell* and $\mathcal{S}$ its *delimiter*. If a vertex $\mathbf{z}$ belongs to the interior facet of $\mathcal{B}$, then the *safe region* of $\mathbf{z}$ with respect to $\mathcal{B}$ is the region inside the delimiter. Otherwise, the *safe region* of $\mathbf{z}$ with respect to $\mathcal{B}$ is the region outside the delimiter. The intersection of the safe regions of $\mathbf{z}$ with respect to each one of its adjacent bi-cells is the *safe region* of $\mathbf{z}$. Finally, if all the bi-cells of $\mathcal{T}$ are safe bi-cells, then we call $\mathcal{T}$ a *safe triangulation*. When a triangulation is a safe triangulation, we say that it verifies the *safety condition*.

It is clear that a safe triangulation is equivalent to a Delaunay triangulation, since:

- Each delimiter can be shrunk in such a way that it touches the vertices of the interior facet, and thus defining an empty-sphere passing through the interior facet of its bi-cell.

- The *empty-sphere* property of Delaunay triangulation facets defines itself an empty-sphere passing through the interior facets of the bi-cells. Those empty-spheres are delimiters.

Proposition 1 *Given a Delaunay triangulation $\mathcal{T}$, if its vertices move arbitrarily yet inside their safe regions, then $\mathcal{T}$ remains Delaunay.*

It is a direct consequence of the equivalence between safe and Delaunay triangulations, since if the vertices remains inside their safe regions, then $\mathcal{T}$ remains a safe triangulation, and hence a Delaunay triangulation.

Among all possible delimiters of a bi-cell, we define the *standard delimiter* as the median hypersphere of the d-annulus with the inner-hypersphere passing through the interior facet and the outer-hypersphere passing through the opposing vertices. Both median hypersphere and d-annulus are unique. We call the d-annulus, the *generator of the standard delimiter*.

Let $\mathcal{D}(\mathcal{B})$ be the delimiter of a given bi-cell $\mathcal{B}$. Then, for a given vertex $\mathbf{z} \in \mathcal{T}$, we define:

$$\tilde{\epsilon}(\mathbf{z}) = \inf_{\substack{\mathcal{B} \ni \mathbf{z} \\ \mathcal{B} \in \mathcal{T}}} dist_H(\mathbf{z}, \mathcal{D}(\mathcal{B})). \tag{3}$$

We have that $\tilde{\epsilon}(\mathbf{z}) \leq \epsilon(\mathbf{z})$, since the delimiter generated by the minimum-width d-annulus of the vertices of a bi-cell $\mathcal{B}$ maximizes the minimum distance of the vertices to the delimiter. If we consider the standard delimiter of a bi-cell as its delimiter, we have that $\tilde{\epsilon}(\mathbf{z}) = \epsilon(\mathbf{z})$.

We define the *tolerance region* of $\mathbf{z}$ as the ball centered at the location of $\mathbf{z}$ with radius $\epsilon(\mathbf{z})$. That is the biggest ball centered at the location of $\mathbf{z}$ and contained inside its safe region. We can always extend the tolerance region to be the entire safe region, but its shape is substantially more complex as it is defined by the intersection of several hyperspheres and complement of hyperspheres. See Figure 1 for an illustration. Details on the computations of such objects can be found in the extended version of this paper [3].

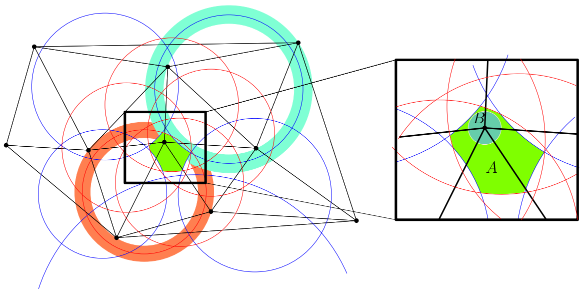


Figure 1: $\mathbf{z} \in \mathbb{R}^2$ the center of B. The region A is the safe region of $\mathbf{z}$, while B is its tolerance region.

3 Delaunay Maintenance Algorithms

Another naive updating algorithm, significantly different from rebuilding, is the *placement* algorithm. It consists of, iterating over all relocated vertices, taking each vertex and walking to the cell containing its new position, inserting a vertex at the new position, and, finally removing the old vertex from the triangulation. Since the cost of deletions is very high, in practice, rebuilding the whole triangulation is faster. However, there is a number of algorithms which require to compute the next location of a vertex one by one, updating the Delaunay triangulation after each relocation. Placement algorithm is dynamic, unlike rebuilding, and it remains suitable for such applications. Another relevant side effect of the static nature of rebuilding is that some significant overhead is necessary to preserve a certain order on accessing the vertices after a relocation, which may be useful for applications that reference the vertices of the triangulation externally and use the displacement algorithm as a black-box.

We redesigned the placement algorithm so as to take into account the tolerance region of each relocated vertex. In practice, the algorithm proposed is capable of correctly decide whether a vertex displacement requires an update of the connectivity or not, so as to trigger the trivial update condition. It will be denoted by *tolerance algorithm*. It is dynamic, preserving all benefits from placement compared to rebuilding.

Data structure. Consider a triangulation $\mathcal{T}$, where for each vertex $\mathbf{z} \in \mathcal{T}$ we associate two point locations: $\mathbf{f_z}$

and $\mathbf{m_z}$. We will call them the *fixed position* and the *moving position* of a vertex. The fixed position will be useful to *fix* a reference position for a moving point. The moving position of a given vertex is its actual position, and will change at any time it is relocated. Initially, the fixed positions and moving positions are equal. We call $\mathcal{T}_\mathbf{f}$ and $\mathcal{T}_\mathbf{m}$ the embedding of $\mathcal{T}$ with respect to $\mathbf{f_z}$ and $\mathbf{m_z}$ respectively. For each vertex, we store two numbers: $\epsilon_\mathbf{z}$ and $D_\mathbf{z}$ (as in **D**isplacement). These numbers represent the tolerance value of $\mathbf{z}$ and the distance between $\mathbf{f_z}$ and $\mathbf{m_z}$ respectively.

Pre-computations. Compute the Delaunay triangulation $\mathcal{T}$ of the initial set of points S, and for each vertex, let $\epsilon_\mathbf{z} = \epsilon(\mathbf{z})$ and $D_\mathbf{z} = 0$. The *updating algorithm* performs as follows for each vertex displacement:

```
Input: A triangulation $\mathcal{T}$ after the pre-computations, a vertex $\mathbf{z}$
       of $\mathcal{T}$ and its new location $\mathbf{p}$.
Output: $\mathcal{T}$ updated after the relocation of $\mathbf{z}$.
$(\mathbf{m_z}, D_\mathbf{z}) \leftarrow (\mathbf{p}, dist(\mathbf{f_z}, \mathbf{p}))$ ;
if $D_\mathbf{z} < \epsilon_\mathbf{z}$ then we are done;
else
    insert $\mathbf{z}$ on a queue $\mathcal{Q}$;
    while $\mathcal{Q}$ is not empty do
        let $\mathbf{h}$ be the head of $\mathcal{Q}$;
        $(\mathbf{f_h}, \epsilon_\mathbf{h}, D_\mathbf{h}) \leftarrow (\mathbf{m_h}, \infty, 0)$ ;
        move $\mathbf{h}$ with placement algorithm;
        foreach new created bi-cells $\mathcal{B}$ do
            $\epsilon' \leftarrow$ half the width of the standard delimiter
            generator of $\mathcal{B}$ ;
            foreach vertex $\mathbf{w} \in \mathcal{B}$ do
                if $\epsilon_\mathbf{w} > \epsilon'$ then
                    $\epsilon_\mathbf{w} \leftarrow \epsilon'$;
                    if $\epsilon_\mathbf{w} < D_\mathbf{w}$ then insert $\mathbf{w}$ into $\mathcal{Q}$;
                end
            end
        end
    end
end
```

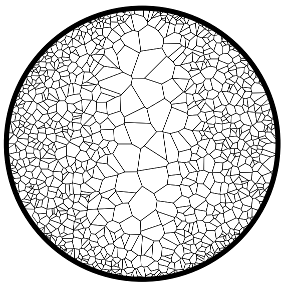
The algorithm is shown to terminate as each processed vertex $\mathbf{z}$ gets a new displacement value $D_\mathbf{z}$ equals to 0 and thus smaller or equal to $\epsilon_\mathbf{z}$. At the end of this algorithm, all vertices are guaranteed to have their $D_\mathbf{z}$ smaller or equal to their $\epsilon_\mathbf{z}$. In such a situation, from Property 1, $\mathcal{T}_\mathbf{m}$ is the Delaunay triangulation of the moving positions. The tolerance algorithm has the same complexity as the placement algorithm. If all points move, the total number of calls to the placement algorithm is guaranteed to be smaller than $O(n)$. Discussion on robustness issues can be found in the extended version of this paper [3].

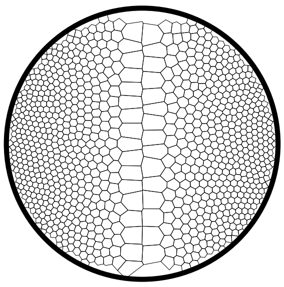
4 Experimental Results

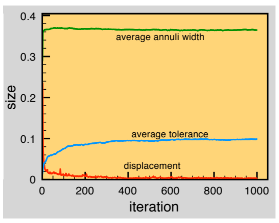
A centroidal Voronoi tessellation is a Voronoi tessellation whose generating points are the centroids (centers of mass) of the corresponding Voronoi regions [6]. Applications of centroidal Voronoi tessellation include image compression, quadrature, and cellular biology, to name a few. There are several approaches to determine centroidal Voronoi tessellations, classified as either probabilistic or deterministic. One deterministic method is the well-known *Lloyd's iterations* [7]. Given a set of points, Lloyd's iterations optimize their placement by moving them to the centroid of their Voronoi region with respect to a given density function, up to convergence. For each iteration of Lloyd's method, we must recompute the Delaunay triangulation of the points. Each iteration can be considered as a distinct timestamp.

Hereafter, we denote the set of the width of the standard delimiter generator of the bi-cells by $\mathcal{W}$, the set of the vertex tolerances by $\epsilon(V)$. The average of a set S of numbers is denoted by $avg(S)$, and the tolerance algorithm by *T*.

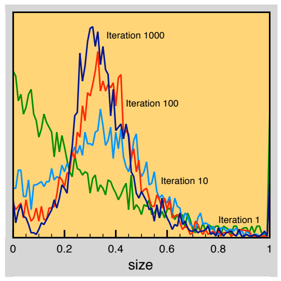
In two dimensions, we consider four different density functions: $\rho_1 = 1$, $\rho_2 = x^2 + y^2$, $\rho_3 = x^2$ and $\rho_4 = \sin^2 \sqrt{x^2 + y^2}$. And we run the Lloyd's iterations to obtain centroidal Voronoi tessellations according to them. Our experiments show that during the process, while the average displacement is going to zero, $avg(\mathcal{W})$ and $avg(\epsilon(V))$ converge to around $33-40\%$ and $5-10\%$ of the average point density respectively. We run the Lloyd's iterations during 1000 iterations, which is required to generate satisfactory results (see Figure 2).


(a) ρ_3 iteration 1.


(b) ρ_3 iteration 1000.



(c) ρ_3 tolerances evolution.



(d) ρ_3 distribution of $\mathcal{W}$.

Figure 2: 1.000 points in a circle with densities ρ_1 and ρ_3.

In three dimensions, we run the Lloyd's iterations on a point set of about 13.000 points in a ball. This experiment is referenced to as *slloyd*. The average tolerance of bi-cells remains about 30% of the point density, but due to the higher degree of a vertex in three dimensions, the average tolerance of the vertices converge to a much smaller value (of about 1%).

We also run a dual version of the Lloyd's algorithm, first called *optimal Delaunay triangulation* (ODT), which is shown to generate fewer *slivers* (flat tetrahedra, which impacts negatively on the stability of computations in simulations) [9]. As this algorithm requires moving the points one by one in sequence, we cannot use the rebuilding scheme. We run it on a set of about 13.000 points in a sphere (*snodt*) and on a surface mesh of a human body of about 8.000 points (*man*). Figure 3 shows *man* at different iterations of the process. Close-ups on the head visually indicates that 100 iterations are clearly not sufficient to get an high quality mesh (as confirmed by less visual quality measures).

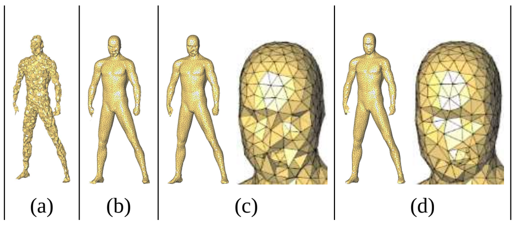


Figure 3: In (a) man initially; In (b), (c) and (d) man after 10, 100 and 1000 iterations respectively. (c) and (d), show close-ups on the head.

Experiments show a good convergence of the average tolerance of bi-cells to a reasonable value and of the average tolerance of vertices to a smaller value, although these tolerances converge slower than in two dimensions.

For a random point distribution and random displacements of length δ, if displacements are small enough, and only around 25% of the vertices have displacement above tolerance in two dimensions (55% in three dimensions), then *T* is competitive with rebuilding. In three dimensions, placement is twice slower and around nine time slower than rebuilding in two and three dimensions respectively. In extreme configurations, our algorithm outperforms rebuilding by a factor of 17 (125 in three dimensions). However those configurations are artificial and very unlikely to happen for real data sets.

The performance of *T* depends on the amount of displacements remaining inside the tolerance region. An idea of what this percentage represents in terms of distances can be shown within the context of random walk. When displacement magnitudes are around 20% of $avg(\epsilon(V))$ of the input set, *T* is still able to outperform rebuilding in both two and three dimensions. In three dimensions, if all displacements magnitudes are lower or equal to $avg(\epsilon(V))$, *T* is more than three time faster than placement.

We now discuss the performance of the algorithms for Lloyd iterations, considering a thousand of iterations. We implemented in addition a small variation of the tolerance algorithm, suggested in Section 3, which consists of rebuilding for the first few iterations, and, swapping to *T*. We denote this algorithm by *R*+*T*.

In two dimensions, *T* and *R*+*T* outperform both rebuilding and placement. In three dimensions, they outperform placement in all configurations. This enhancement is relevant for applications requiring to move vertices one at a time and for dynamic triangulations, which cannot be achieved by rebuilding. However, to outperform rebuilding is harder in three dimensions, because the removal operation is even slower and the number of bi-cells containing a given point is five times larger than in two dimensions. In spite of that, *T* still outperforms rebuilding in synthetic and real data sets ($snodt$ and man). *T* cannot perform with $slloyd$, as well as it does with the other inputs, because of the *slivers* produced by running a pure Lloyd iteration (tolerances are sensitive to *slivers*). In both two and three dimensions, when we go further on the number of iterations, *T* becomes faster. As shown in Figure 3, the number of iterations highly impacts the quality of the final result. This result enables a novel possibility: Going further on the number of iterations. More details on experimental results are available in the full version of the paper [3].

5 Conclusion

This paper deals with the problem of updating Delaunay triangulations for moving points. We introduce the notion of the tolerance and safe region of a vertex in this context. We end up with an algorithm suitable when the magnitude of the displacement keeps decreasing while the tolerances keep increasing. Such configurations translate into convergent schemes, e.g. Lloyd's iterations itself.

References

[1] Alliez P., Cohen-Steiner D., Yvinec M., and Desbrun M. Variational tetrahedral meshing. *ACM Trans. on Graphics*, 24:617–625, 2005. SIGGRAPH '2005 Conf. Proc.

[2] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. In *Proc. 8th ACM-SIAM Sympos. Disc. Algo.*, pages 528–537, January 1997.

[3] De Castro P. M. M. and Devillers O. Delaunay triangulations for moving points. Research report, INRIA, 2008, http://hal.inria.fr/inria-00344053/.

[4] Debard J. B., Balp R., and Chaine R. Dynamic Delaunay Tetrahedralisation of a Deforming Surface. *The Visual Computer*, page 12 pp, Aug. 2007.

[5] Devillers O. Improved incremental randomized Delaunay triangulation. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 106–115, 1998.

[6] Du Q., Faber V., and Gunzburger M.. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Rev.*, 41(4):637–676, 1999.

[7] Ostrovsky R., Rabani Y., Leonard J. Schulman, and Swamy C. The effectiveness of Lloyd-type methods for the k-means problem. In *FOCS '06: Proc. of the 47th Annu. IEEE Sympos. on Found. of Comp. Sci.*, pages 165–176, Washington, DC, USA, 2006. IEEE Computer Society.

[8] D. Russel. *Kinetic Data Structures in Practice*. PhD thesis, Stanford Univ., 2007.

[9] Tournois J., Alliez P., Wormser C., and Desbrun M. Quality isotropic tetrahedron meshing with optimal Delaunay triangulations, 2008.

Linear-Time Delaunay Triangulations Simplified*

Kevin Buchin† Wolfgang Mulzer‡

Abstract

Recently it was shown that — under reasonable assumptions — Voronoi diagrams and Delaunay triangulations of planar point sets can be computed in time $o(n \log n)$, beating the classical comparison-based lower bound. A number of increasingly faster randomized algorithms have been proposed, most recently a linear-time algorithm based on a randomized incremental construction that uses a combination of nearest neighbor graphs and the history structure to speed up point location. We present a simpler variant of this approach relying only on nearest neighbor graphs. The algorithm and its analysis generalize to higher dimensions, with an expected performance that is proportional to the structural change of the randomized incremental construction. As a by-product, we analyze an interesting class of insertion orders for randomized incremental constructions.

1 Introduction

A classical lower bound asserts that it takes $\Omega(n \log n)$ time to construct the Delaunay triangulation (DT) of a planar point set in the algebraic decision tree model. However, three years ago Chan and Pătraşcu [5] showed that this lower bound can be broken on a word RAM, giving a randomized algorithm that runs in $O(n \log n / \log \log n)$ time, which they later improved to $n 2^{O(\sqrt{\log \log n})}$ [6]. More recently, a randomized linear-time algorithm in a different model was proposed [2]. In this paper, we show how to simplify this approach by using a variant of a randomized incremental construction (RIC) con BRIO.

Our main tool is a reduction from nearest neighbor graphs (NNGs) to Delaunay triangulations: we show that if the NNG for an m-point set can be computed in time $f(m)$, then the Delaunay triangulation for an n-point set P can be computed in time $O(C(n) + f(n))$, where $C(n)$ is the expected structural change for a RIC. In a recent paper, Chan [4] explains a linear-time reduction from quadtrees to NNGs and describes two settings in which quadtrees can be constructed quickly. The first assumes a real-RAM with a constant-time *restricted* floor-function that can be applied only if the resulting integer has $O(\log n)$ bits. Thus, we avoid issues about creating an unreasonably powerful model of computation. Furthermore, we assume that the input P has polynomially bounded *spread* (defined below). Then, the NNG of P, $\mathrm{N}(P)$, can be computed in linear time [4]. In the second setting, we have a word RAM which can compute the *shuffle* of a point in constant time (if the coordinates of p are $(p_{1w} \cdots p_{11}, p_{2w} \cdots p_{21}, \ldots, p_{dw} \cdots p_{d1})$, the shuffle is $p_{1w} p_{2w} \cdots p_{dw} \cdots p_{d1}$). This is a reasonable assumption, since the shuffle operation is in AC^0. Here, a NNG can be found in the time needed for integer sorting. We believe that current integer sorting algorithms can be adapted to the shuffle order and that therefore the assumption can be dropped.

Our reduction follows the RIC paradigm, choosing a random permutation of the input and inserting the points into the DT one by one. However, unlike the previous algorithm [2], the choice is not uniform, but we use a sampling strategy that can be seen as a generalization of a biased randomized insertion order (BRIO) [1]. Compared to the classical approach, RIC con BRIO achieves an improved locality of reference as follows: take a *gradation* $P = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_\ell$, where $|S_\ell| = O(1)$ and S_{i+1} is obtained from S_i by sampling each point independently with probability 1/2. When performing the RIC, first insert all points in S_ℓ, then $S_{\ell-1}$, then $S_{\ell-2}$, and so on. The order within the sets S_i is arbitrary; eg, it could be optimized for the underlying hardware. This procedure runs in expected time proportional to what classical RICs achieve [1, 3].

Owing to the flexibility of RICs, our algorithm works in any dimension. For the planar case, we can give a very simple analysis for the reduction. However, this analysis crucially uses the fact that the worst-case complexity of planar DTs is linear. Of course, this is no longer true in higher dimensions, but even then, "typical" point sets often have linear-size DTs. Even stronger, it is not uncommon to encounter inputs in which the size of the DT for a random subset is linear in the subset. For example, this happens for points that are distributed uniformly in a d-dimensional ball. For such point sets the structural change of classical RICs and RICs con BRIO is linear (but they lose a logarithmic factor for updating the

*The first author was supported by the Netherlands' Organisation for Scientific Research (NWO) under BRICKS/FOCUS project no. 642.065.503. The second author was supported in part by NSF grant CCF-0634958 and NSF CCF 0832797.

†Dept. of Information and Computing Sciences, Utrecht University, Netherlands, buchin@cs.uu.nl

‡Department of Computer Science, Princeton University, wmulzer@cs.princeton.edu

conflict information). As for many random distributions, the expected spread is polynomially bounded, which suffices to compute the NNG in linear time in our first model of computation. In a more refined analysis, we will show that the expected running time of our reduction is proportional to the expected structural change of the RIC plus the time for computing the NNG. Thus, on typical inputs we again achieve an improvement in running time over classical RICs.

2 Algorithm & First Analysis

We say that an n-point set $P \subseteq \mathbb{R}^d$ has polynomially bounded *spread*, if the ratio between the maximum and minimum distance between any two points in P is $O(n^c)$, for some constant c. NNGs for point sets with bounded spread or for points with integer coordinates can be computed quickly, as was shown by Chan [4].

Theorem 1 (Chan [4]) *Let $P \subseteq \mathbb{R}^d$ be an n-point set. If P has polynomially bounded spread,* $\mathrm{N}(P)$ *can be computed in time $O(n)$ on a real-RAM with the restricted floor function. If the coordinates of the points are b-bit integers and we assume a word RAM with the shuffle operation, then* $\mathrm{N}(P)$ *can be found in time $O(\mathrm{sort}(n))$, where $\mathrm{sort}(n)$ is the time for sorting db-bit integers.*

We now describe our algorithm `BrioDC`. Suppose that NNGs can be computed in $f(m)$ time, such that $f(m)/m$ is increasing. By Theorem 1, we have $f(n) = O(n)$ or $f(n) = O(\mathrm{sort}(n))$, depending on the model. Given a point set P, we first compute $\mathrm{N}(P)$ in $O(f(n))$ time. Next we compute a sample $S \subseteq P$ such that S meets every connected component of $\mathrm{N}(P)$ and such that S contains every point in P with probability 1/2. This is done as follows: we define a (partial) matching $\mathcal{M}(P)$ on P by pairing up two arbitrary points in each component of $\mathrm{N}(P)$. The sample S is obtained by picking one random point from each pair in $\mathcal{M}(P)$ and sampling the points in $P \setminus \mathcal{M}(P)$ independently with probability 1/2 (although they could also be paired up). We recursively compute the Delaunay triangulation $\mathrm{DT}(S)$ of S. To locate $P \setminus S$ in $\mathrm{DT}(S)$, we traverse the components of $\mathrm{N}(P)$ starting in each component at a point of S and inserting points while we walk through the Delaunay triangulation along the edges of $\mathrm{N}(P)$. Since $f(m)/m$ is increasing, it takes $O(f(n))$ time to compute the NNGs for all the samples.

Theorem 2 *Let P be a planar n-point set.* `BrioDC` *computes* $\mathrm{DT}(P)$ *in expected time $O(f(n)+n)$, where $f(m)$ is the time to compute a m-point NNG and we assume that $f(m)/m$ is increasing.*

Proof. The cost of a straight-line walk along an edge pq of $\mathrm{N}(P)$ with p in the current DT and q to be inserted next can be split into the cost of finding the triangle at p in which the walk starts and the cost of the actual traversal. The first part of the cost can be bounded by the number of triangles at p in the current DT. Summing over all walking steps, any triangle is counted for each of its vertices at most as often as the degree of this vertex in $\mathrm{N}(P)$, which is bounded. The second part of the cost, ie, the cost of the traversal, can be bounded by the structural change since all triangles traversed are destroyed when the next point is inserted.

The structural change can be bounded by comparing it to the structural change of the last round of a RIC con BRIO. Let p_s be the probability that a given triangle with s conflicts appears in the last round of such a construction. We have $p_s = c/2^s$ for a suitable constant c. The probability p'_s of this triangle appearing in `BrioDC` while constructing $\mathrm{DT}(P)$ from $\mathrm{DT}(S)$ is also bounded by $1/2^s$: either the stoppers of the triangle are sampled independently of each other (then we directly get this bound), or not (then S includes a stopper and the simplex cannot occur). Thus, the expected structural change is asymptotically as for RIC con BRIO and therefore linear [1, 3]. Now, the expected size of S is $|P|/2$, and we can apply the argument above to the construction of $DT(S)$, and so on. Overall this yields the desired running time. $\square$

3 Dependent Insertion Orders & Second Analysis

Theorem 2 does not apply to higher dimensions, since it requires that the *worst-case* complexity of a planar DT is linear. We now make the analysis sensitive to the expected structural change of a RIC.

Theorem 3 *Let P be a d-dimensional n-point set. The expected running time of* `BrioDC` *is $O(C(P) + f(n))$, where $C(P)$ denotes the expected structural change incurred by a RIC on P and f is as in Theorem 2. The constant in the O-notation depends exponentially on d.*

Let $P = S_0 \supseteq \cdots \supseteq S_\ell$ be the sequence of samples taken by `BrioDC`. Fix a set $\mathbf{u}$ of $d+1$ distinct points in P. Let Δ be the simplex spanned by $\mathbf{u}$, and let $L_\mathbf{u} \subseteq P$ denote the points in the circumsphere of Δ. The set $\mathbf{u}$ is the *trigger* set, $L_\mathbf{u}$ the *stopper* set for Δ. Consider the event A_α that Δ occurs in one of the DTs in the construction of $\mathrm{DT}(S_\alpha)$ from $\mathrm{DT}(S_{\alpha+1})$, for some α. Clearly, A_α can only happen if $\mathbf{u} \subseteq S_\alpha$ and $L_\mathbf{u} \cap S_{\alpha+1} = \emptyset$.

We will prove Theorem 3 using Lemma 4 which in turn follows from Proposition 6.

Lemma 4 *We have*

$$\Pr[A_\alpha] \le e^{2d+2}\, 2^{-(d+1)\alpha} \left(1 - 2^{-\alpha-1}\right)^{|L_\mathbf{u}|}.$$

Let us first give some intuition: we think of the sampling as an $(\alpha+1)$-step random walk on a path with $|L_{\mathbf{u}}|$ nodes: a random step represents a new sample, and each node represents the current number of stoppers. The goal is to upperbound the probability of reaching state 0 while retaining all triggers. This model is overly simplistic: the random choices in a step not only depend on the number of stoppers in the current sample S, but also on the matching $\mathcal{M}(S)$. Even worse, the probability distribution in the current state may depend on past states. However, we will show how to avoid these issues through appropriate conditioning, and that the random walk essentially behaves like a Markov process that in each round eliminates $d+1$ stoppers and samples the remaining stoppers independently. The elimination is due to trigger-stopper pairs, since we assume that all triggers survive. The remaining stoppers are not necessarily independent, but matching two stoppers guarantees that one of them survives, which can only help. Eliminating $d+1$ stoppers in the ith round has a similar effect as starting with about $(d+1)2^i$ fewer stoppers: though a given trigger can be matched with only one stopper per round, these parings can vary for different instances of the walk, and since a given stopper survives a round with probability roughly 1/2, the "amount" of stoppers eliminated by one trigger in all instances roughly doubles per round.

Proof. (of Lemma 4) For a sample $S \subseteq P$, we define the *matching profile* for S as the triple (a, b, c) that counts the number of trigger-stopper, stopper-stopper, and trigger-trigger pairs in $\mathcal{M}(S)$. In order to bound $\Pr[A_\alpha]$, we consider

$$p_{s,k} := \max_{\mathcal{P}_k} \Pr[A_\alpha \mid X_{s,k}, \mathcal{P}_k], \tag{1}$$

where we define

$$X_{s,k} := \{\mathbf{u} \subseteq S_{\alpha-k}\} \cap \{|L_{\mathbf{u}} \cap S_{\alpha-k}| = s\},$$

ie, the event that the sample $S_{\alpha-k}$ contains all the triggers and exactly s stoppers. The maximum in (1) is taken over all possible sequences $\mathcal{P}_k = \mathbf{m}_0, \dots, \mathbf{m}_{\alpha-k-1}, Y_0, \dots, Y_{\alpha-k-1}$ of matching profiles $\mathbf{m}_i$ for S_i and events Y_i of the form $X_{t_i,\alpha-i}$ for some t_i. Since $\Pr[A_\alpha] = p_{|L_{\mathbf{u}}|,\alpha}$, it suffices to upperbound $p_{s,k}$. We define a recursion for $p_{s,k}$. To do that, let

$$T_k := \{\mathbf{u} \subseteq S_{\alpha-k}\},$$

ie, the event that $S_{\alpha-k}$ contains all the triggers, and let

$$U_{k,i} := \{|L_{\mathbf{u}} \cap S_{\alpha-k}| = i\},$$

denote the event that $S_{\alpha-k}$ contains exactly i stoppers.

Proposition 5 *We have*

$$p_{s,k} \le \max_{\boldsymbol{m}} \Pr[T_{k-1} \mid X_{s,k}, \boldsymbol{m}] \cdot \sum_{i=0}^{s} p_{i,k-1} \Pr[U_{k-1,i} \mid T_{k-1}, X_{s,k}, \boldsymbol{m}],$$

where the maximum is over all possible matching profiles $\boldsymbol{m} = (a, b, c)$ for $S_{\alpha-k}$.

Proof. Fix a sequence $\mathcal{P}_k$ as in (1). Then, by distinguishing how many stoppers are present in $S_{\alpha-k+1}$,

$$\Pr[A_\alpha \mid X_{s,k}, \mathcal{P}_k] = \sum_{i=0}^{s} \Pr[X_{i,k-1} \mid X_{s,k}, \mathcal{P}_k] \Pr[A_\alpha \mid X_{i,k-1}, X_{s,k}, \mathcal{P}_k].$$

Now if we condition on a matching profile $\mathbf{m}$ for $S_{\alpha-k}$, we get

$$\Pr[X_{i,k-1} \mid X_{s,k}, \mathcal{P}_k, \mathbf{m}] = \Pr[T_{k-1} \mid X_{s,k}, \mathbf{m}] \Pr[U_{k-1,i} \mid T_{k-1}, X_{s,k}, \mathbf{m}],$$

since the distribution of triggers and stoppers in $S_{\alpha-k+1}$ becomes independent of $\mathcal{P}_k$ once we know the matching profile and the number of triggers and stoppers in $S_{\alpha-k}$. Furthermore,

$$\Pr[A_\alpha \mid X_{i,k-1}, \mathbf{m}, X_{s,k}, \mathcal{P}_k] \le \max_{\mathcal{P}_{k+1}} \Pr[A_\alpha \mid X_{i,k-1}, \mathcal{P}_{k+1}] = p_{i,k-1}.$$

The claim follows by taking the maximum over $\mathbf{m}$. $\square$

We use Proposition 5 to bound $p_{s,k}$: if $\mathbf{m} = (a, b, c)$ pairs up two triggers, we get $\mathbf{u} \not\subseteq S_{\alpha-k+1}$ and $\Pr[T_{k-1} \mid X_{s,k}, \mathbf{m}] = 0$. Hence we can assume $c = 0$ and therefore $\Pr[T_{k-1} \mid X_{s,k}, \mathbf{m}] = 1/2^{d+1}$, since all triggers are sampled independently. Furthermore, we know that none of the a stoppers paired with a trigger and half of the $2b$ stoppers paired with a stopper end up in $S_{\alpha-k+1}$, while the remaining $t_{\mathbf{m}} := s - a - 2b$ stoppers are sampled independently. Thus, Proposition 5 gives

$$p_{s,k} \le \max_{\substack{\mathbf{m} \\ c=0}} \sum_{i=b}^{s-a-b} \frac{p_{i,k-1}}{2^{d+1}} \Pr\left[B_{1/2}^{t_{\mathbf{m}}} = i - b\right], \tag{2}$$

where $B_{1/2}^{t_{\mathbf{m}}}$ denotes a binomial distribution with $t_{\mathbf{m}}$ trials and success probability 1/2.

Proposition 6 *We have*

$$p_{s,k} \le 2^{-(d+1)k} \left(1 - 2^{-k-1}\right)^s \prod_{j=1}^{k} \left(1 - 2^{-j}\right)^{-d-1}.$$

Proof. The proof is by induction on k. For $k = 0$, we have

$$p_{s,0} \le \left(1 - \frac{1}{2}\right)^s,$$

since we require that none of the s stoppers in S_α be present in $S_{\alpha+1}$, and this can only happen if they are sampled independently of each other. Furthermore, by (2),

$$\begin{aligned} & p_{s,k+1} \\ & \le \max_{\substack{\mathbf{m} \\ c=0}} \sum_{i=b}^{s-a-b} \frac{p_{i,k}}{2^{d+1}} \Pr\left[B_{1/2}^{t_\mathbf{m}} = i - b\right] \qquad (3) \\ & = \max_{\substack{\mathbf{m} \\ c=0}} \frac{1}{2^{d+1+t_\mathbf{m}}} \sum_{i=0}^{t_\mathbf{m}} \binom{t_\mathbf{m}}{i} p_{i+b,k}. \end{aligned}$$

Using the inductive hypothesis and the binomial theorem, we bound the sum as

$$\begin{aligned} & \sum_{i=0}^{t_\mathbf{m}} \binom{t_\mathbf{m}}{i} p_{i+b,k} \\ & \le \frac{\sum_{i=0}^{t_\mathbf{m}} \binom{t_\mathbf{m}}{i} \left(1 - 2^{-k-1}\right)^{i+b}}{2^{(d+1)k}} \prod_{j=1}^{k} \left(1 - 2^{-j}\right)^{-d-1} \\ & = \frac{\left(2 - 2^{-k-1}\right)^{t_\mathbf{m}}}{2^{(d+1)k}} \left(1 - 2^{-k-1}\right)^b \prod_{j=1}^{k} \left(1 - 2^{-j}\right)^{-d-1} \\ & = \frac{\left(1 - 2^{-k-2}\right)^{t_\mathbf{m}}}{2^{(d+1)k - t_\mathbf{m}}} \left(1 - 2^{-k-1}\right)^b \prod_{j=1}^{k} \left(1 - 2^{-j}\right)^{-d-1}. \end{aligned}$$

Now, since $t_\mathbf{m} = s - a - 2b \ge s - d - 1 - 2b$ and since

$$\frac{\left(1 - 2^{-k-1}\right)^b}{\left(1 - 2^{-k-2}\right)^{2b}} = \left(\frac{1 - 2^{-k-1}}{1 - 2^{-k-1} + 2^{-2k-4}}\right)^b \le 1,$$

it follows that

$$\begin{aligned} & \sum_{i=0}^{t_\mathbf{m}} \binom{t_\mathbf{m}}{i} p_{i+b,k} \\ & \le \frac{\left(1 - 2^{-k-2}\right)^s}{2^{(d+1)k - t_\mathbf{m}}} \prod_{j=1}^{k+1} \left(1 - 2^{-j}\right)^{-d-1}, \end{aligned}$$

and hence (3) gives

$$p_{s,k+1} \le \frac{\left(1 - 2^{-k-2}\right)^s}{2^{(d+1)(k+1)}} \prod_{j=1}^{k+1} \left(1 - 2^{-j}\right)^{-d-1},$$

which finishes the induction. □

(Proof of Lemma 4 continued) Now, since $(1 - x) > \exp(-x/(1 - x))$ for $x < 1$ we have

$$\prod_{j=1}^{k} (1 - 2^{-j})^{-d-1} \le e^{2(d+1)\sum_{j=1}^{\infty} 2^{-j}} = e^{2(d+1)},$$

so we get as claimed

$$\Pr[A_\alpha] \le p_{|L_\mathbf{u}|,\alpha} \le e^{2d+2} 2^{-(d+1)\alpha} (1 - 2^{-\alpha-1})^{|L_\mathbf{u}|}.$$

□

Proof. (of Theorem 3) The cost for NNG computations and walking can be bounded as in the proof of Theorem 2 (in particular the degree of the NNG is bounded by a constant for any fixed dimension), except that the expected structural change is not necessarily linear. To prove the theorem, it is sufficient to show that the probability that the simplex Δ spanned by $\mathbf{u} \subseteq P$ occurs in `BrioDC` is up to a constant factor upper bounded by the same probability in a RIC. In the case of `BrioDC`, this probability is bounded by $\sum_{\alpha=0}^{\infty} \Pr[A_\alpha]$. Let p denote the corresponding probability in a RIC and let

$$p_\alpha := 2^{-(d+1)\alpha} (1 - 2^{-\alpha-1})^{|L_\mathbf{u}|} (1 - 2^{-d-1}).$$

We have

$$\Pr[A_\alpha] \le \exp(2d + 2)(1 - 2^{-d-1})^{-1} p_\alpha.$$

Now, from an analysis of RIC con BRIO [3, Lemma 3.8] we have, $\sum_{\alpha=0}^{\infty} p_\alpha \le 2^{d+1} p$, thus,

$$\sum_{\alpha=0}^{\infty} \Pr[A_\alpha] \le 2^{d+1} \exp(2d + 2)(1 - 2^{-d-1})^{-1} p.$$

□

References

[1] N. Amenta, S. Choi, and G. Rote. Incremental constructions con BRIO. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 211–219. ACM Press, 2003.

[2] K. Buchin. Delaunay triangulations in linear time? Submitted, see `arXiv:0812.0387v1 [cs.CG]`.

[3] K. Buchin. *Organizing Point Sets: Space-Filling Curves, Delaunay Tessellations of Random Point Sets, and Flow Complexes.* PhD thesis, Free University Berlin, 2007. http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000003494.

[4] T. M. Chan. Well-separated pair decomposition in linear time? *Inform. Process. Lett.*, 107(5):138–141, 2008.

[5] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. *SIAM J. Comput.* To appear. Preliminary versions in: Proc. 47th IEEE Sympos. Found. Comput. Sci. (FOCS), 2006, pp. 325–332, 333–342.

[6] T. M. Chan and M. Pătraşcu. Voronoi diagrams in $n \cdot 2^{O(\sqrt{\lg \lg n})}$ time. In *Proc. 39th Annu. ACM Sympos. Theory Comput.*, pages 31–39, 2007.

2D Delaunay mesh generation with area/aspect-ratio constraints

Narcís Coll* Marité Guerrieri* J. Antoni Sellarès*

Abstract

In this paper we propose algorithms that combine Delaunay approaches with control area/aspect-ratio constraints for generating a refined Delaunay mesh of a Planar Straight Line Graph domain. We also present a series of applications which combine area and aspect-ratio constraints with angle constraint. This combination is possible because all the designed algorithms are based on Delaunay triangulation.

1 Introduction

Many works exist on the generation of a quality triangular mesh for a Planar Straight Line Graph (PSLG) domain. There are several quality criteria, being the most common that the angles of each triangle are neither too small nor too large. Delaunay refinement mesh generation algorithms have taken place in this context (to only cite a few references, see [5, 4, 3, 7, 1, 8]).

Some problems require a certain granularity (average triangle area) of the mesh, instead of an angle restriction. However, applying area constraints without taking into account the angles is not ideal for mesh generation because, in most cases, the resulting mesh can have many long and skinny triangles. Combining the application of an area constraint to regions of the mesh with a later angle refinement allows the adaptation of a domain to many physics problems. Moreover, a triangulation with triangles of area close to equal has the important property that the distribution of its vertices is very uniform [6, 2] and can help later insertion of new elements of the PSLG. In this context we devote our efforts to study area based constraints, having the combination of Delaunay criterion and an improvement process in mind.

Following this line of thinking, it is natural to continue with the study of the behavior of an aspect-ratio quality measure based on area instead of one developed for the angle criterion. By using this measure we avoid triangles with relatively small areas. When using the aspect-ratio quality measure, a local improvement of the triangulation can be obtained by flipping a diagonal of a convex quadrilateral. Nevertheless, the sequence of local improvements along the whole mesh does not produce an unique triangulation. To assure an unique triangulation we use a Delaunay method in combination with the aspect-ratio constraint.

Our approach is focused in optimizing the position of new or existing Steiner vertices. The optimal position is looked for in the star defined by a vertex of a bad element. The new position can not change its link. If the link changes all the work carried out to optimize the position of the point is lost and link changes can produce very bad triangles. The link is maintained by the definition of *Delaunay zones* introduced in the next section.

2 General notations and Delaunay zones

Let $\mathcal{T}$ a conforming Delaunay triangulation of a given PSLG Ω. Being q a vertex of $\mathcal{T}$ and e an edge of the link $\mathcal{L}_q$ of q, we use the following notation:

- $H_{q,e}$ is the open half-plane determined by e and containing the vertex q.
- $t_{q,e}$ is the triangle with vertices e_i, e_f and q.
- $t'_{q,e}$ is the adjacent triangle to $t_{q,e}$ by e.
- $A_{q,e}$ is the area of $t_{q,e}$.
- $c_{q,e}$ is the circumcircle of $t'_{q,e}$ when e is not constrained and it is the *diametral lens* of 30° angle otherwise.
- $a_{q,e}$ is the arc $c_{q,e} \cap H_{q,e}$. We will say that $c_{q,e}$ is the supporting circle of $a_{q,e}$.
- $\mathcal{S}_q$ is the *star* of a vertex q.
- $ker(\mathcal{L}_q)$ is the *kernel* of $\mathcal{L}_q$.

The *Delaunay zone of an edge* $e \in \mathcal{L}_q$, denoted $\mathcal{D}_{q,e}$, is the set of points of $H_{q,e}$ external to $c_{q,e}$.

The *external Delaunay zone of a vertex* q is the set $\mathcal{ED}_q = \bigcap_{e \in \mathcal{L}_q} \mathcal{D}_{q,e}$. The external Delaunay zone of a vertex is an open non-convex set included in $ker(\mathcal{L}_q)$ and may be constituted by several non-connected components.

Let $C_{\mathcal{L}_q}$ be the convex vertices of $\mathcal{L}_q$. The *Delaunay zone of a vertex* $u \in C_{\mathcal{L}_q}$, denoted $\mathcal{D}_{q,u}$, is the interior of the circle determined by u and its adjacent vertices in $\mathcal{L}_q$. The *internal Delaunay zone of a vertex* q is the convex set $\mathcal{ID}_q = \bigcap_{u \in C_{\mathcal{L}_q}} \mathcal{D}_{q,u}$.

The *Delaunay zone of a vertex* q is the set $\mathcal{D}_q = \mathcal{ED}_q \cap \mathcal{ID}_q$. The boundary of $\mathcal{D}_q$ will be denoted by $\overline{\mathcal{D}_q}$. If we replace q by a point $p \in \mathcal{D}_q$ then $\mathcal{L}_p = \mathcal{L}_q$

*Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain. Email: {coll,mariteg,sellares}@ima.udg.edu. Partially supported by the Spanish Ministerio de Educación y Ciencia under grant TIN2007-67982-C02-02.

and the set of triangles $\{t_{q,e}\}$ is substituted by the set $\{t_{p,e}\}$.

3 Finding a point optimizing a Delaunay zone

Consider a vertex q of $\mathcal{T}$. We are interested in replacing the point q by a point $\widetilde{p} \in D_q$, to assure $\mathcal{L}_p = \mathcal{L}_q$, such that optimizes diverse criteria related to the triangles of the star $\mathcal{S}_q$. We assume that the triangulation $\mathcal{T}$ is located on the XY-plane of $\mathbb{R}^3$ and the edges of any star $\mathcal{S}_q$ are ordered counterclockwise.

3.1 Area constraint

Considering the objectives of achieving a triangulation where the maximum area of the triangles is lower than a certain preestablished bound α, and the number of vertices added to the triangulation is as low as possible, we optimize the position of a new or a existing Steiner vertex in its Delaunay zone.

The area $A_{p,e}$ of a triangle $t_{p,e}$ is a linear function of $p \in \mathcal{S}_q$, denoted $\pi_e(p)$, whose graph is (a bounded part) of a plane π_e through the segment e. Let Π_q be the the set of all planes π_e, $e \in \mathcal{L}_q$. Then we have:

$$\min_{e\in\mathcal{L}_q}\{A_{p,e}\} = \min_{\pi_e\in\Pi_q}\{\pi_e(p)\} = \mathcal{LE}_{\Pi_q}(p) \qquad \text{and}$$

$$\max_{e\in\mathcal{L}_q}\{A_{p,e}\} = \max_{\pi_e\in\Pi_q}\{\pi_e(p)\} = \mathcal{UE}_{\Pi_q}(p).$$

3.1.1 Optimization criteria

We want to find $\widetilde{p} \in \mathcal{D}_q$ that satisfies one of the following criteria:

- $\widetilde{p}$ maximizes the minimum area among the triangles of $\mathcal{S}_q$:

$$\max_{p\in\mathcal{D}_q}(\min_{e\in\mathcal{L}_q}\{A_{p,e}\}) = \max_{p\in\mathcal{D}_q}\mathcal{LE}_{\Pi_q}(p) = \mathcal{LE}_{\Pi_q}(\widetilde{p}).$$

This means to find the highest point of $\mathcal{LE}_{\Pi_q}(p)$.

- $\widetilde{p}$ minimizes the difference between the maximum and minimum area among the triangles of $\mathcal{S}_q$:

$$\min_{p\in\mathcal{D}_q}(\max_{e\in\mathcal{L}_q} A_{p,e} - \min_{e\in\mathcal{L}_q} A_{p,e}) = \min_{p\in\mathcal{D}_q}(\mathcal{UE}_{\Pi_q}(p) - \mathcal{LE}_{\Pi_q}(p)) =$$
$$= \mathcal{UE}_{\Pi_q}(\widetilde{p}) - \mathcal{LE}_{\Pi_q}(\widetilde{p}).$$

This means finding the lowest point of $\mathcal{UE}_{\Pi_q}(p) - \mathcal{LE}_{\Pi_q}(p)$.
The difference $\mathcal{UE}_{\Pi_q}(p) - \mathcal{LE}_{\Pi_q}(p)$ can be computed as the upper envelope of another set of planes in the following way. For each two different edges e, e' of $\mathcal{L}_q$ let $\pi_e - \pi_{e'}$ be the plane determined by the graph of the linear function $A_{p,e} - A_{p,e'}$. Let $D\Pi_q$ be the set of planes $\{\pi_e - \pi_{e'} \ / \ e, e' \in \mathcal{L}_q, e \neq e'\}$. Then, we have: $\mathcal{UE}_{\Pi_q}(p) - \mathcal{LE}_{\Pi_q}(p) = \mathcal{UE}_{D\Pi_q}(p)$, and, consequently:

$$\min_{p\in\mathcal{D}_q}(\mathcal{UE}_{\Pi_q}(p) - \mathcal{LE}_{\Pi_q}(p)) = \min_{p\in\mathcal{D}_q}(\mathcal{UE}_{D\Pi_q}(p)) = \mathcal{UE}_{D\Pi_q}(\widetilde{p}).$$

3.2 Aspect-Ratio constraint

The aspect-ratio $asp(t)$ of a triangle t is defined as the ratio between the area of the triangle and the square of its longest edge. Given a fixed aspect-ratio ϵ our goal it to find a conforming Delaunay triangulation $\mathcal{T}$ of Ω satisfying

$$\forall t \in \mathcal{T} \ asp(t) \geq \epsilon \,.$$

Given a half-edge h with length $||h||$ of $\mathcal{T}$ we define the *aspect-ratio* of h with respect to a point p as

$$asp(h,p) = \frac{A_{p,h}}{||h||^2} \,.$$

Observe that function $asp(h,p)$ is linear with respect to p. Let v_h be the vertex of the triangle of $\mathcal{T}$ at the left and adjacent to h. Since the *aspect-ratio* of a triangle t bounded by three half-edges h_0, h_1, h_2 can be computed by

$$asp(t) = \min\{asp(h_0, v_{h_0}), asp(h_1, v_{h_1}), asp(h_2, v_{h_2})\}\,,$$

our goal is equivalent to find a conforming Delaunay triangulation $\mathcal{T}$ of Ω satisfying

$$\forall h \in \mathcal{T} \ asp(h, v_h) \geq \epsilon \,.$$

Our approach is based on a local maximization of the aspect-ratios of the link half-edges of the vertices. Then, a half-edge having an aspect-ratio less than ϵ is called a *bad half-edge*.

3.2.1 Aspect-ratio link maximization

We are interested in replacing a vertex q by a point $\widetilde{p} \in D_q$ such that maximizes the aspect ratio of the half-edges $h \in \mathcal{L}_q$. Then, we want to find the point $\widetilde{p} \in D_q$ that satisfies:

$$\min_{h\in\mathcal{L}_q}\{asp(h,\widetilde{p})\} = \max_{p\in\mathcal{D}_q}\{\min_{h\in\mathcal{L}_q}\{asp(h,p)\}\}\,.$$

The linear function $asp(h,p)$ of $p \in D_q$, denoted $\pi_h(p)$, has a plane π_h through the half-edge h as graph. Consequently, we have

$$\min_{h\in\mathcal{L}_q}\{asp(h,p)\} = \mathcal{LE}_{\Pi_q}(p)$$

where Π_q is the the set of all planes π_h, $h \in \mathcal{L}_q$. Then, point $\widetilde{p}$ corresponds to the highest point of $\mathcal{LE}_{\Pi_q}(p)$.

3.3 Optimization algorithms

Our method first finds the local extremum p^* that optimizes the areas/aspect-ratios in $\mathcal{S}_q$. If $p^* \in \mathcal{D}_q$ then p^* is the optimal solution $\widetilde{p}$. Otherwise we find the point $\overline{p}$ that optimizes the areas/aspect-ratios restricted to $\overline{\mathcal{D}_q}$ and we take a point inside $\mathcal{D}_q$ close to $\overline{p}$

as the target point $\widetilde{p}$. This process needs to compute the arcs of $\overline{\mathcal{D}_q}$ and the envelopes $\mathcal{LE}_{\Pi_q}$ and $\mathcal{UE}_{D\Pi_q}$.

Since the computation depends on the number of edges of $\mathcal{L}_q$ whose average number is six, the time needed for the overall method is constant.

In the following we describe in more detail the two steps to find p^* and, if it is the case, $\widetilde{p}$, described in the previous paragraph.

First step. The first step finds a point p^* on $\mathcal{S}_q$ satisfying one of the criteria established.

• **Finding a point maximizing the minimum area or the minimum aspect-ratio.** The local extremum $p^* = (x^*, y^*)$ can be found by computing the $\mathcal{LE}_{\Pi_q}$ and searching its maximum with a cost of $\mathcal{O}(k \log k)$, being k the number of $\mathcal{L}_q$ edges. Nevertheless, the point p^* can be optimally computed by applying linear programming to the optimization problem that maximizes function $\Theta(x, y, z) = z$ subject to restrictions $\{z \leq \pi_e(x, y) \mid e \in \mathcal{L}_q\}$ in $\mathcal{O}(k)$ time.

• **Finding a point minimizing the difference between areas.** The local extremum $p^* = (x^*, y^*)$ can be found as the minimum of the $\mathcal{UE}_{D\Pi_q}$ with a cost of $\mathcal{O}(k^2 \log k)$, being k the number of $\mathcal{L}_q$ half-edges. Again, the point p^* can be optimally computed by applying linear programming to the optimization problem that minimizes function $\Theta(x, y, z, t) = z$ subject to restrictions $\{z \leq \pi_e(x, y), z + t \geq \pi_e(x, y) \mid e \in \mathcal{L}_q\}$ in $\mathcal{O}(k)$ time.

Second step. The second step finds a point $\widetilde{p}$ on $\overline{\mathcal{D}_q}$. Let $\overline{F}$ be the set of the projected faces of the polyhedral surface considered ($\mathcal{LE}_{\Pi_q}$ or $\mathcal{UE}_{D\Pi_q}$). The point $\widetilde{p}$ is found by applying the following steps:

1. Compute the arcs a_i of $\overline{\mathcal{D}_q}$.
2. Split each arc a_i in subarcs a_{ij} according to the faces of $\overline{F}$ intersected by a_i.
3. For each subarc a_{ij} compute the global extremum point p_{ij} of the linear function associated to the face intersected by a_{ij}.
4. Let $\overline{p}$ be the point p_{ij} that optimizes the criterion considered and $\widetilde{p}$ be a point inside $\mathcal{D}_q$ close to $\overline{p}$.

4 Generating a Delaunay refined mesh using area or aspect-ratio constraint

The process to generate a refined Delaunay quality mesh of a PSLG considering area or aspect-ratio constraint consists on the following steps. First, a conforming Delaunay triangulation of the PSLG is generated, then the list of bad triangles (area $\geq \alpha$) or bad half-edges (aspect-ratio $\geq \epsilon$) to be removed is obtained, and finally our improvement method is applied to eliminate the bad elements.

Improvement process. The improvement process receives as input the list of bad elements to be removed. The output of the process is a mesh with the desired quality. The process actualizes this list after the treatment of each bad triangle or bad half-edge and finishes when it is empty.
To remove a bad triangle using an area criterion, first its Steiner vertices are moved to a better position. This process of movement finishes when all three points were checked for movement or when the triangle has a good area. If, after the previous movement process the triangle continues being a bad one, the midpoint of its longest edge is inserted and moved to an optimal position.
To remove a bad half-edge using an aspect-ratio criterion, first the opposite vertex to its longest edge is checked for deletion or movement to optimal position in order to achieve a triangle with good aspect-ratio. If this is not possible or the half-edge continues with a bad aspect-ratio then the midpoint of its LEPP [4] is inserted and moved to an optimal position. The deletion operation used is the same basic operation explained in [1].

Movement of Steiner vertices. We name *restricted* Steiner vertices those located on any segment of the PSLG and *free* Steiner vertices the remaining Steiner ones. A free vertex q is substituted by the best point $\widetilde{p}$ in D_q such that it optimizes the area/aspect-ratio criterion. The movement of a restricted vertex is constrained over their correspondent subsegment. The optimizing algorithm can easily be adapted in order to guarantee that the vertex $\widetilde{p}$ lies on the subsegment.

5 Combining different constraints

Combination of area and aspect-ratio constraints with the angle criterion exposed in [1] is possible because all the methods proposed are based on Delaunay triangulation.
Increasing granularity. Combining the application of an area constraint to regions of the mesh with a later angle refinement allows the automatic adaptation of the domain to physics problems (see Figure 1).

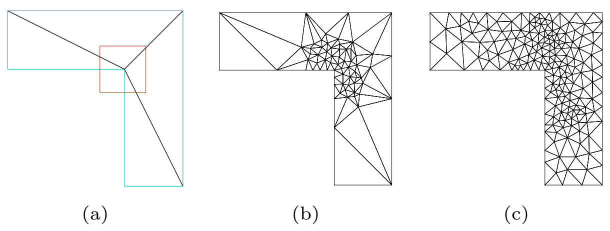


Figure 1: Boundary into the red square in (a) must have more granularity than in the rest of the model. In (b) the resulting mesh after applying a preprocess in which triangles into the square have area less than 0.01. In (c) all the angles are greater than 40° after applying the angle criterion.

When the granularity is increased for the whole domain better results in the number of final elements are obtained than if only the angle criterion is applied (see Table 1 and Figure 2).

Model	Area criterion	α	Triangles preprocess	Triangles angle $\geq 40°$
Superior Lake	Maximizing	0.1	1234	4176
Airfoil	minimum	4.0	173	706
Superior Lake	Minimizing	0.1	1084	4661
Airfoil	difference	4.0	150	622

Table 1: Results obtained applying the area criteria as a preprocess to obtain triangles with angles greater o equal to 40°. Applying only the angle criterion with 40° bound the number of triangles are: 4693 for the Superior Lake model and 773 for the airfoil model.

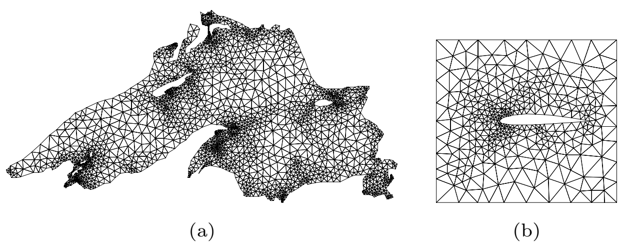


Figure 2: Triangulations with all the angles greater or equal to 40° after a preprocess using an area constraint with a bound of 0.1. (a) Superior Lake. (b) Airfoil.

Preprocess for domain changes. The application of a preprocess, considering area constraints, facilitates the posterior insertion of new restrictions to the domain because of the adequate granularity produced. In Figure 3 first the Superior Lake without the islands has been triangulated with an area constrain, second the islands has been inserted, and third a triangulation with all the angles greater or equal to 40° has been obtained.

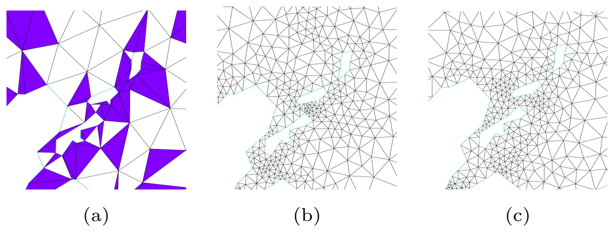


Figure 3: A detail of three islands in the Superior Lake model. (a) Insertion of the islands. Triangles with minimum angle less than 40° are in blue. (b) Triangulation with all the angles greater or equal than 40°. The total number of triangles obtained applying this strategy is 4109 which is less than 4693 the quantity obtained applying directly the minimum angle criterion in (c).

Aspect-ratio and angle constraints. Applying the aspect-ratio quality measure a good distribution of minimum angles is obtained (see Table 2). The triangulation of Figure 4(a) has been obtained using an aspect-ratio constraint of 0.175 bound. It has 624 triangles, 22.27° minimum angle, 40.23° mean angle and 35.31° first quartile angle. If the angle criterion with bound 40° is applied later, the triangulation of Figure 4(b) is obtained with 2415 triangles. Applying only the angle criterion with a bound of 40° a triangulation with 2660 triangles would have been obtained.

Model	Mean angle	1st quartile	Minimum angle
Superior Lake	40.73	35.25	21.43
Airfoil	39.83	33.35	22.81

Table 2: Mean angle, first quartile, and minimum angle obtained by applying the aspect-ratio constraint with 0.175 bound.

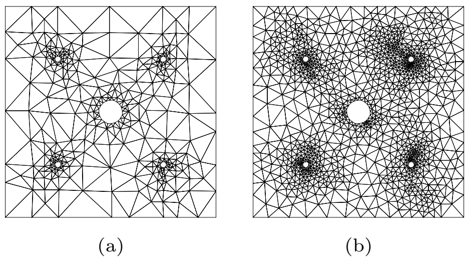


Figure 4: (a) Triangulation obtained using an aspect-ratio constraint of 0.175. (b) Triangulation with all the angles greater or equal to 40° applying the angle criterion to the triangulation (a).

References

[1] N. Coll, M. Guerrieri, and J.-A. Sellarès. Combining improvement and refinement techniques: 2d delaunay mesh adaptation under domain changes. *Applied Mathematics and Computation*, 201:527–546, 2008.

[2] L. Freitag, M. Jones, and P. Plassmann. An efficient parallel algorithm for mesh smoothing. *In Proceedings of the 4th International Meshing Roundtable*, 47–58, 1995.

[3] S.-E. Pav. Delaunay Refinement Algorithms. *Department of Mathematical Sciences - Carnegie Mellon University - PhD thesis*, 2003.

[4] M-C. Rivara, N. Hitschfeld, and R-B. Simpson. Terminal edges delaunay (small angle based) algorithm for the quality triangulation problem. *Computer-Aided Design*, 33:263–277, 2001.

[5] J.-R. Shewchuk. Delaunay Refinement Mesh Generation. *School of Computer Science - Carnegie Mellon University - PhD thesis*, 1997.

[6] V. Surazhsky and C. Gotsman. High quality compatible triangulations. *Engineering with Computers*, 20(2):147–156, 2004.

[7] J. Tournois, P. Alliez, and O. Devillers. Interleaving Delaunay Refinement and Optimization for 2D Triangle Mesh Generation *In Proceedings of the 16th International Meshing Roundtable*, 83–101, 2007.

[8] A. Üngör Off-centers: A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. *Computational Geometry: Theory and Applications*, 42(2):109–118, 2009.

Homotopic Rectilinear Routing with Few Links and Thick Edges[*]

Bettina Speckmann[†] Kevin Verbeek[†]

Abstract

We study the problem of finding non-crossing thick minimum-link rectilinear paths homotopic to a set of input paths in an environment with rectangular obstacles. This problem occurs in the context of map schematization under geometric embedding restrictions, for example, when schematizing a highway network for use as a thematic layer. We present a 2-approximation algorithm that runs in $O(n^3 + k_{in} \log n + k_{out})$ time, where n is the total number of input paths and obstacles and k_{in} and k_{out} are the total complexities of the input and output paths, respectively. Our algorithm not only approximates the minimum number of links, but also minimizes the total length of the paths. An approximation factor of 2 is optimal when using smallest paths as lower bound.

1 Introduction

Motivation. Schematic maps are a well-known cartographic tool; they visualize a set of nodes and edges (for example, highway or metro networks) in simplified form to communicate connectivity information as effective as possible. Many schematic maps are somewhat removed from the geographic context: while the locations of nodes are usually close to their actual geographic location, edges can be routed in any way that is consistent with the network topology. However, if one wishes to use a schematized network as a thematic layer for a thematic map, then additional geographic and hence geometric restrictions apply. Consider the following example: we want to create a thematic map that focuses on traffic flow on highways. As base map we use a standard geographic map, the thematic overlay consists of a schematized highway network. The schematized network has to fulfill various requirements: (*i*) edges are drawn thickly, using few orientations and links, (*ii*) critical features (cities, lakes, etc.) of the base map are not obscured and retain their correct topological position with respect to the network. There has been little algorithmic work on network schematization under geometric embedding restrictions; in this paper we address one of the fundamental underlying problems for the first time.

Our input is a set of n_o rectangular obstacles and n_p pairs of points (a_i, b_i) together with non-crossing paths π_i that connect a_i to b_i. We want to find a set of n_p non-crossing rectilinear thick a_i-b_i paths which are homotopic to the paths π_i. We refer to this problem as the *thick routing problem* (correspondingly, the *thin routing problem* is the same problem for standard "thin" paths). Returning to the example above, the obstacles model the (bounding boxes of) critical map features and the paths π_i correspond to the actual geographic location of the highways we want to schematize. Note that the endpoints of the paths—the cities connected by the highways—necessarily also constitute obstacles when considering homotopy.

Related Work. The thick routing problem can be seen as a variation on the *thick non-crossing paths problem* studied by Mitchell and Polishchuk [11]. They find shortest non-crossing thick paths in a polygonal domain, that is, the points a_i and b_i lie on the boundary of a simple polygon. We consider general input paths, albeit with fixed homotopy classes, and study minimum-link rectilinear instead of shortest paths. There are several papers [2, 6] that find shortest paths homotopic to a given collection of input paths. However, while a set of shortest paths homotopic to a set of non-crossing input paths is necessarily non-crossing, the same does not hold for minimum-link rectilinear paths. Our problem is also related to *drawing graphs with fat edges* [5], and to *wire routing* in VLSI design [4, 7, 9, 10], although none of these papers strives to minimize the number of links.

Many variants of the thick routing problem, even without obstacles and with thin paths, are proven NP-hard by Bastert and Fekete [1] if the homotopy classes of the paths are not specified. Yang *et al.* [14] consider a very restricted version: they show how to find a pair of non-crossing minimum-link rectilinear paths inside a rectilinear polygon. Pach and Wenger [13] prove that non-crossing paths connecting n pairs of points might need $\Omega(n)$ paths to have $\Omega(n)$ links. Gupta and Wenger [8] present an approximation algorithm (with an approximation factor larger than 120) for finding non-crossing minimum-link paths inside a simple polygon; here all endpoints lie on the boundary of the polygon. Another type of related work considers map schematization and metro map construction. For example, Cabello *et al.* [3] give an algorithm that schematizes a network using 2 or 3 links per path, if possible. Nöllenburg and Wolff [12]

[*]Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707.

[†]Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands, speckman@win.tue.nl and k.a.b.verbeek@tue.nl

use a method based on mixed-integer programming to generate metro maps using one edge per path. Both methods do not incorporate obstacles and are restricted to a small constant number of links per path.

Results. We present a 2-approximation algorithm for the thick routing problem which runs in $O(n^3 + k_{in} \log n + k_{out})$ time, where $n = n_o + n_p$ is the total number of obstacles and input paths and k_{in} and k_{out} are the total complexities of the input and output paths, respectively. As a lower bound for the minimum number of links any solution must have we use the total number of links of the *smallest paths* that are homotopic to the input paths (a smallest path is a rectilinear path that is both shortest and minimum-link). Our algorithm not only approximates the minimum number of links, but also minimizes the total length of the paths. An approximation factor of 2 is optimal when using smallest paths as lower bound.

Our algorithm is based on a surprisingly simple incremental construction, which we first explain for thin paths in Section 3, before extending it to thick paths in Section 4. Intuitively, any incremental approach to this problem should be doomed due to a cascading number of links, but we show how to move already inserted paths without increasing the number of links or creating crossings to make room for each new path. To efficiently move homotopic (sub-)paths we use a bundling technique as described in [6]. Furthermore, we "grow" the thick paths by extending the results of [5] to rectilinear paths and rectangular obstacles.

2 Preliminaries

Every rectilinear path π consists of a sequence of horizontal and vertical links. We can also see a rectilinear path as a sequence of staircase chains separated by U-turns. We distinguish two types of staircase chains: *positive staircase chains* which go right and down (or left and up) and *negative staircase chains* which go left and down (or right and up). If there is an obstacle on the inside of a U-turn touching the middle link, then we call this U-turn a *tight U-turn.*

A *smallest path* is a rectilinear path that is both shortest and minimum-link. Every staircase chain between two points has the same length. Hence a rectilinear path is shortest iff it has only tight U-turns. Every collection of non-crossing rectilinear paths can be made shortest by making the U-turns tight one-by-one. So we can find a solution to the thin (or thick) routing problem that is also shortest. However, rectilinear shortest paths are not unique and can cross. Therefore we consider *rectilowest paths*, which are rectilinear shortest paths of which all staircase chains are as low as possible. *Rectihighest paths* are defined similarly. We can show that rectilowest paths or rectihighest paths are always non-crossing. Unfortunately they can have $O(n)$ times more links than smallest paths, so we use smallest paths as a starting point for the algorithm instead. We observe that a positive and a negative staircase chain of rectilinear shortest paths can never cross.

Consider a collection of non-crossing rectilinear y-monotone chains π_i and a collection of obstacles ω_i. We can represent the homotopy classes of π_i by a total order $\mathcal{O}$ on the paths and obstacles. Use $\mathcal{O}(\pi_i)$ for the position of π_i in this order. Let π'_i and π'_j be paths homotopic to π_i and π_j ($\pi'_i \sim_h \pi_i$ and $\pi'_j \sim_h \pi_j$). If $\mathcal{O}(\pi_i) < \mathcal{O}(\pi_j)$, then an *intersection region* of π'_i and π'_j is a region enclosed by π'_i and π'_j at y-coordinates where the paths are out of order, i.e. where π'_j is to the left of π'_i. Intersection regions are y-monotone rectilinear polygons and do not contain obstacles.

3 Thin Paths

We solve the thin routing problem in two steps. First we compute the smallest paths homotopic to the input paths π_i. Then we untangle these paths.

Computing smallest paths. We compute *pushed* smallest paths that are homotopic to the input paths. A *pushed* rectilinear shortest path has all horizontal links pushed down as much as possible (while keeping it shortest) and all vertical links pushed right (left) as much as possible for positive (negative) staircase chains. Pushed smallest paths have only rectangular intersection regions, which simplifies the untangling algorithm. To compute them efficiently, we first compute rectilowest and rectihighest paths. For that we modify the algorithm by Efrat *et al.* [6], which also bundles homotopic y-monotone chains. The rectilowest and rectihighest paths are represented by $O(n)$ bundles with at most $O(n)$ links each. From that we can easily compute a bundled representation of the pushed smallest paths using a plane sweep algorithm. In total this step takes $O(n^2 + k_{in} \log n)$ time.

Untangling smallest paths. To untangle the smallest paths, we use the fact that positive staircase chains do not cross negative staircase chains. Hence we can restrict ourselves to positive staircase chains. As these chains have a total order, we assume the input paths to be ordered positive staircase chains π_i ($1 \leq i \leq n$).

We untangle the paths incrementally, adding paths in order from left to right. The first two paths π_1 and π_2 can have only rectangular intersection re-

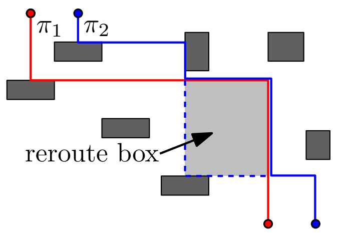


Figure 1: Untangling the first two paths.

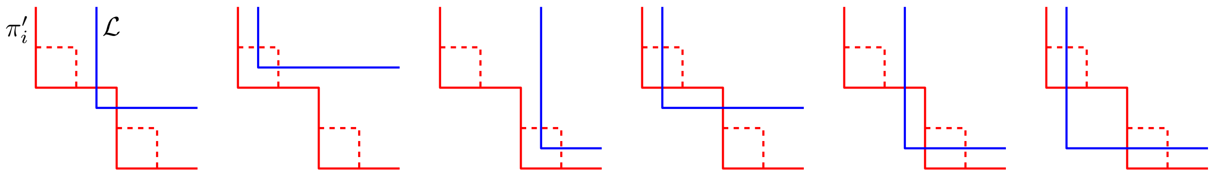


Figure 2: The possible shapes of intersection regions of $\mathcal{L}$ and a path π'_i. Dashed lines denote reroute boxes.

gions. We can add two links to π_2 (for each intersection region) to remove the crossings. Unfortunately this might make intersection regions with π_k ($k > 2$) non-rectangular such that rerouting requires more links. To avoid this we keep track of where paths are rerouted using *reroute boxes* (Fig. 1). Reroute boxes are rectangular and contain no obstacles.

We maintain the following invariants after adding path π_k (rerouted paths are denoted by π'_i): (i) $\pi'_1 \dots \pi'_k$ are non-crossing, (ii) $\pi_1 \dots \pi_k$ have at most one reroute box per lower-left bend, and (iii) all upper-right bends of a path π_j are to the right of π'_i with $i < j$. A new path π_{k+1} is added one L-segment $\mathcal{L}$ at a time. $\mathcal{L}$ can cross multiple other paths. The original paths π_i have only rectangular intersection regions. $\mathcal{L}$ is unchanged and we have added only reroute boxes to $\pi'_1 \dots \pi'_k$, hence there are only a few shapes possible for the intersection regions with $\mathcal{L}$ (Fig. 2): one, two, or three upper-right bends.

Now assume that $\mathcal{L}$ is crossed by the paths $\pi'_a \dots \pi'_b$ ($a \leq b$). We want to make the intersection region of $\mathcal{L}$ with π'_b rectangular such that we can add a reroute box to $\mathcal{L}$ to remove all crossings. We do this incrementally for all intersection regions, starting with π'_a and ending with π'_b. Assume an intersection region $\mathcal{R}_i$ of $\mathcal{L}$ and $\pi'_i \in [\pi'_a \dots \pi'_b]$ is rectangular. To make the intersection region $\mathcal{R}_{i+1}$ rectangular, we move the links of π'_{i+1} through $\mathcal{R}_{i+1}$ (intersection regions contain no obstacles), but not through $\mathcal{R}_i$ (this would introduce crossings with π'_i). By Invariant (i) π'_{i+1} does not cross $\mathcal{R}_i$ initially. If $\mathcal{R}_{i+1}$ contains one upper-right bend, we do nothing as $\mathcal{R}_{i+1}$ is already rectangular. If $\mathcal{R}_{i+1}$ contains two upper-right bends, then we either move the first vertical link of π'_{i+1} to the left (onto $\mathcal{L}$) or the last horizontal link down. Due to the rectangular shape of $\mathcal{R}_i$, we can always do one of the two moves without crossing $\mathcal{R}_i$. If $\mathcal{R}_{i+1}$ contains three upper-right bends, then either $\mathcal{R}_i$ is in the middle corner and we move the first vertical link of π'_{i+1} to the left and the last horizontal link down, or $\mathcal{R}_i$ is not in the middle corner, in which case we can simplify the middle corner and handle this as a case with two upper-right bends (see Fig. 3 for an example).

Since the algorithm maintains all invariants, the output paths are non-crossing. They are also homotopic to the input paths, because we change paths only by adding reroute boxes or by moving links through intersection regions. We add only two links per reroute box and there is at most one reroute box per lower-left bend. Because a positive staircase chain with L links has at most $L/2$ lower-left bends, the algorithm at most doubles the number of links.

We can easily extend this algorithm to work on the y-monotone chains of the smallest paths. Because we can add an L-segment in $O(n)$ time and we have only $O(n)$ chains with at most $O(n)$ links each, untangling the chains can be done in $O(n^3)$ time. Finally we can unbundle the paths in $O(k_{out})$ time. So in total we can compute a 2-approximation for the thin routing problem in $O(n^3 + k_{in} \log n + k_{out})$ time.

4 Thick Paths

In this section we extend the algorithm of the previous section to thick paths. For $\Delta > 0$ let $\mathcal{S}_\Delta$ be the square of size Δ centered at the origin. A *thick rectilinear path* with *spine* π_i and thickness Δ_i is defined as the Minkowski sum of π_i and $\mathcal{S}_{\Delta_i}$: $(\pi_i)^{\Delta_i} = \pi_i \oplus \mathcal{S}_{\Delta_i}$. We say that two paths π_i and π_j have the *proper distance* if $(\pi_i)^{\Delta_i}$ and $(\pi_j)^{\Delta_j}$ are interior disjoint.

The main difference with thin paths is that fixing a thick path might make it impossible for another thick path to be routed. We say that a path $\pi'_i \sim_h \pi_i$ is *feasible* if there exist paths $\pi'_j \sim_h \pi_j$ for $1 \leq j \leq n_p$ and $i \neq j$ such that all paths have the proper distance. In a valid solution of the thick routing problem every path is feasible. Hence we use feasible smallest paths as a lower bound.

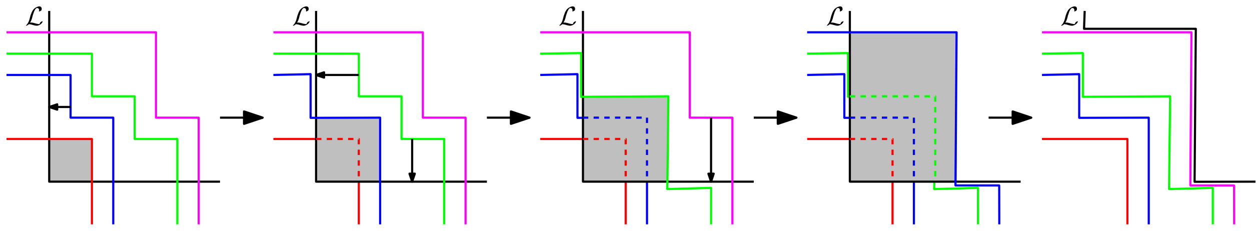


Figure 3: Making intersection regions rectangular and rerouting $\mathcal{L}$.

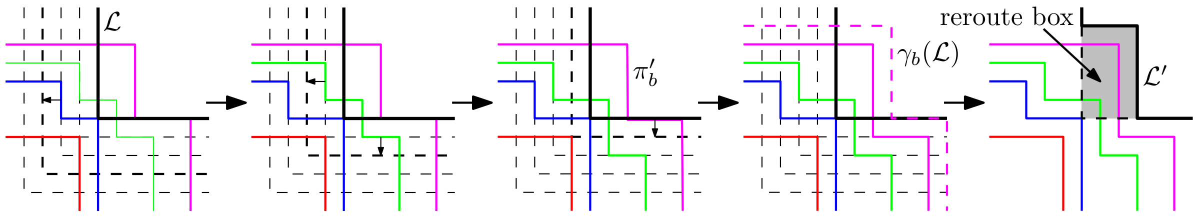


Figure 4: Making guarded regions rectangular.

Feasible smallest paths. To compute feasible smallest paths, we first compute the feasible rectilowest and rectihighest paths. For this we use a variant of the algorithm by Duncan *et al.* [5]. Like in Section 3, we first compute the thin rectilowest and rectihighest paths. Then, using a similar approach as in [5], we grow the paths until they have the required thickness. By construction the resulting paths are the feasible rectilowest and rectihighest paths. After that it is easy to compute feasible smallest paths. Like for thin paths, we bundle homotopic y-monotone chains, so we get $O(n)$ bundles with at most $O(n)$ links each. This can be computed in $O(n^3 + k_{in} \log n)$ time.

Untangling thick paths. The thick paths need to be untangled in such a way that the resulting paths are feasible and have the proper distance. To achieve this, we use *guards*. A *guard* $\gamma_i(j)$ of a path π_i with respect to another path π_j is a path representing the closest position at which the path π_j can be with respect to π_i while all paths have the proper distance. A guard ignores all obstacles. In other words, the guard $\gamma_i(j)$ defines the region where π_j is feasible, if we fix the path π_i and ignore the obstacles. This way a guard $\gamma_i(j)$ is a shifted version of π_i, where the distance depends on the thickness of the paths between π_i and π_j. This distance can differ along the path π_i.

To ensure that we do not change the homotopy classes during untangling, we use the intersection regions between a guard $\gamma_i(j)$ and a path π_j, which we call *guarded regions*. We can show that also guarded regions contain no obstacles and, if the paths are pushed, all guarded regions are rectangular as well.

We now untangle the paths as before, changing two of the invariants: (i') $\pi'_1 \dots \pi'_k$ have the proper distance and (iii') upper-right bends of a guard $\gamma_j(i)$ are to the right of paths π'_i with $i < j$. When adding an L-segment $\mathcal{L}$ we now look at the guarded regions of $\gamma_{\mathcal{L}}(i)$ and π'_i instead of the ordinary intersection regions. Computing the guards and guarded regions is nontrivial and the details can be found in the full paper. Next, we have to make the guarded regions rectangular, which can be done similarly as for thin paths. While doing this, we push the links of a path π'_i onto $\gamma_{\mathcal{L}}(i)$ instead of $\mathcal{L}$. This way Invariant (i') is maintained by definition of the guards. Finally we can add a reroute box to $\mathcal{L}$, such that $\mathcal{L}$ has the proper distance to the other paths (see Fig. 4).

Theorem 1 *We can compute a 2-approximation for the thick routing problem that also minimizes the lengths of the paths in $O(n^3 + k_{in} \log n + k_{out})$ time.*

References

[1] O. Bastert and S. P. Fekete. Geometrische Verdrahtungsprobleme. Technical Report 247, Mathematisches Institut, Universität zu Köln, 1996.

[2] S. Bespamyatnikh. Computing homotopic shortest paths in the plane. *J. Alg.*, 49(2):284–303, 2003.

[3] S. Cabello, M. de Berg, and M. van Kreveld. Schematization of networks. *Comp. Geom.: Theory and Appl.*, 30(3):223–238, 2005.

[4] R. Cole and A. Siegel. River routing every which way, but loose. In *Proc. 25th Symp. Found. Comp. Sci.*, pp. 65–73, 1984.

[5] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. *Intern. J. Found. Comp. Sci.*, 17(5):1143–1163, 2006.

[6] A. Efrat, S. G. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. *Comp. Geom.: Theory and Appl.*, 35(3):162–172, 2006.

[7] S. Gao, M. Jerrum, M. Kaufmanm, K. Mehlhorn, and W. Rülling. On continuous homotopic one layer routing. In *Proc. 4th Symp. Comp. Geom.*, pp. 392–402, 1988.

[8] H. Gupta and R. Wenger. Constructing pairwise disjoint paths with few links. *ACM Trans. Alg.*, 3(3):26, 2007.

[9] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar vlsi layouts. In *Proc. 17th Symp. Theory Comp.*, pp. 69–78, 1985.

[10] M. Malley. *Single-layer wire routing and compaction.* MIT Press, Cambridge, MA, USA, 1990.

[11] J. S. Mitchell and V. Polishchuk. Thick non-crossing paths and minimum-cost flows in polygonal domains. In *Proc. 23rd Symp. Comp. Geom.*, pp. 56–65, 2007.

[12] M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In *Proc. 13th Intern. Symp. Graph Drawing*, LNCS 3843, pp. 321–333, 2005.

[13] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.

[14] C. D. Yang, D. T. Lee, and C. K. Wong. The smallest pair of noncrossing paths in a rectilinear polygon. *IEEE Trans. Comp.*, 46(8):930–941, 1997.

Area-Universal Rectangular Layouts*

David Eppstein† Elena Mumford‡ Bettina Speckmann‡ Kevin Verbeek‡

Abstract

A rectangular layout is a partition of a rectangle into a finite set of interior-disjoint rectangles. A layout is *area-universal* if any assignment of areas to rectangles can be realized by a combinatorially equivalent rectangular layout. We identify a simple necessary and sufficient condition for a rectangular layout to be area-universal: a rectangular layout is area-universal if and only if it is *one-sided*. More generally, given any rectangular layout $\mathcal{L}$ and any assignment of areas to its regions, we show that there can be at most one layout (up to horizontal and vertical scaling) which is combinatorially equivalent to $\mathcal{L}$ and achieves a given area assignment. We also investigate similar questions for perimeter assignments. The adjacency requirements for the rectangles of a rectangular layout can be specified in various ways, most commonly via the dual graph of the layout. We show how to find an area-universal layout for a given set of adjacency requirements whenever such a layout exists.

1 Introduction

Motivation. Raisz [7] introduced *rectangular cartograms* in 1934 as a way of visualizing spatial information, such as population or economic strength, of a set of regions like countries or states. Rectangular cartograms represent geographic regions by rectangles; the positioning and adjacencies of the rectangles are chosen to suggest their geographic locations, while their areas are chosen to represent the numeric values being communicated by the cartogram.

Often more than one numeric quantity should be displayed as a cartogram for the same set of geographic regions. To make the visual comparison of multiple related cartograms easier, it is desirable that the arrangement of rectangles be combinatorially equivalent in each cartogram, although the relative sizes of the rectangles will differ. This naturally raises the question: when is this possible?

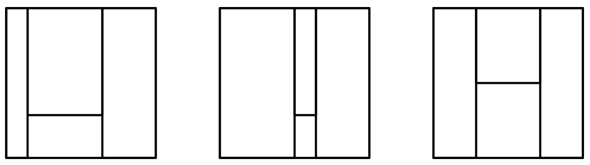


Figure 1: Three area assignments.

Mathematically, a rectangular cartogram is a *rectangular layout*: a partition of a rectangle into finitely many interior-disjoint rectangles. We call a layout $\mathcal{L}$ *area-universal* if, for any area requirement for its regions, some combinatorially equivalent layout $\mathcal{L}'$ has regions with the specified areas. For instance, the four-region rectangular layout shown above with three different area assignments is area-universal: any four numbers can be used as the areas of the rectangles in a combinatorially equivalent layout.

Area-universal rectangular layouts are useful not only for displaying multiple side-by-side cartograms for different sets of data on the same regions, but also for dynamically morphing from one cartogram into another. Additionally, rectangular layouts have other applications in which being able to choose a layout first and then later assigning varying areas while keeping the combinatorial type of the layout fixed may be an advantage: in circuit layout applications of rectangular layouts [10], each component of a circuit may have differing implementations with differing tradeoffs between area, energy use, and speed; and in building design it is desirable to be able to determine the areas of different rooms according to their function [3].

Results. We identify a simple necessary and sufficient condition for a rectangular layout to be area-universal: a rectangular layout is area-universal if and only if it is *one-sided.* One-sided layouts are characterized via their *maximal line segments.* A line segment of a layout $\mathcal{L}$ is formed by a sequence of consecutive inner edges of $\mathcal{L}$. A segment of $\mathcal{L}$ that is not contained in any other segment is maximal. In a one-sided layout every maximal line segment s must be the

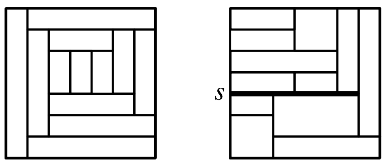


Figure 2: The left layout is one-sided, but the right one is not: s is not the side of any rectangle.

*D.E. is supported in part by NSF grant 0830403 and by the Office of Naval Research under grant N00014-08-1-1015. B.S. and K.V. are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707.

†Department of Computer Science, University of California, Irvine, USA, eppstein@ics.uci.edu

‡Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands, e.mumford@tue.nl, speckman@win.tue.nl, and k.a.b.verbeek@tue.nl

side of at least one rectangle R; any vertices interior to s are T-junctions that all have the same orientation, pointing away from R (Fig. 2). Given an area-universal layout $\mathcal{L}$ and an assignment of areas for its regions, we describe a numerical algorithm that finds a combinatorially equivalent layout $\mathcal{L}'$ whose regions have a close approximation to the specified areas.

More generally, given any rectangular layout $\mathcal{L}$ and any assignment of areas to its regions, we show that there can be at most one layout (up to horizontal and vertical scaling) which is combinatorially equivalent to $\mathcal{L}$ and achieves the given area assignment. This result was previously known only for two special classes of rectangular layouts, namely *sliceable layouts* (layouts that can be obtained by recursively partitioning a rectangle by horizontal and vertical lines) and *L-shape destructable layouts* [9] (layouts where the rectangles can be iteratively removed such that the remaining rectangles form an L-shaped polygon).

We also investigate *perimeter cartograms* in which the perimeter of each rectangle is specified rather than its area. Again, any rectangular layout can have at most one combinatorially equivalent layout for a given perimeter assignment; it is possible in polynomial time to find this equivalent layout, if it exists.

The rectangles of a rectangular cartogram should have the same adjacencies as the regions of the underlying map. Hence, the dual graph of the cartogram should be the same as the dual graph of the map. The dual of a rectangular cartogram or layout must be a triangulated plane graph satisfying certain additional conditions. We call such graphs *proper graphs.* Every proper graph $\mathcal{G}$ has at least one *rectangular dual*: a rectangular layout $\mathcal{L}$ whose dual graph is $\mathcal{G}$. However, not every proper graph has an area-universal rectangular dual; Rinsma [8] described an outerplanar proper graph $\mathcal{G}$ and an assignment of weights to the vertices of $\mathcal{G}$ such that no rectangular dual of $\mathcal{G}$ can have these weights as the areas of its regions. We describe algorithms that, given a proper graph $\mathcal{G}$, find an area-universal rectangular dual of $\mathcal{G}$ if it exists. These algorithms are not fully polynomial, but are fixed-parameter tractable for a parameter related to the number of separating four-cycles in $\mathcal{G}$.

In the following we can only sketch our results, a full version of the paper can be found here [2].

2 Preliminaries

A rectangular layout is a partition of a rectangle into a finite set of interior-disjoint rectangles, where no four regions meet in a single point. We denote the dual graph of a layout $\mathcal{L}$ by $\mathcal{G}(\mathcal{L})$. A layout $\mathcal{L}$ such that $\mathcal{G} = \mathcal{G}(\mathcal{L})$ is called a *rectangular dual* of graph $\mathcal{G}$. $\mathcal{G}(\mathcal{L})$ is a plane triangulated graph and is unique for any layout $\mathcal{L}$. Not every plane triangulated graph has a rectangular dual, and if it does, then the rectangular dual is not necessarily unique. Kozminski and Kinnen [6] proved that a plane triangulated graph $\mathcal{G}$ has a rectangular dual if and only if we can augment $\mathcal{G}$ with four external vertices in such a way that the *extended graph* $E(\mathcal{G})$ has the following two properties: (i) every interior face is a triangle and the exterior face is a quadrangle; (ii) $E(\mathcal{G})$ has no separating triangles. If a plane triangulated graph $\mathcal{G}$ allows such an augmentation, then we say that $\mathcal{G}$ is a *proper graph.* A rectangular dual of an extended graph of a proper graph $\mathcal{G}$ can be constructed in linear time [5] and it immediately implies a rectangular dual for $\mathcal{G}$ (Fig. 3).

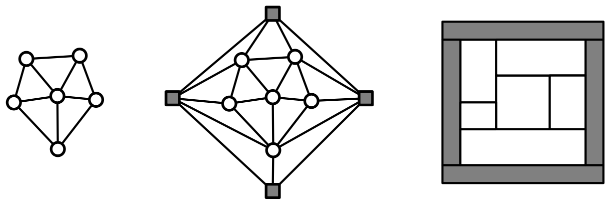


Figure 3: A proper graph $\mathcal{G}$, an extended graph $E(\mathcal{G})$, and a rectangular dual $\mathcal{L}$ of $E(\mathcal{G})$.

An extended graph $E(\mathcal{G})$ determines uniquely which vertices of a proper graph $\mathcal{G}$ are associated with the corner rectangles of every rectangular dual of $\mathcal{G}$ that corresponds to $E(\mathcal{G})$. For a given proper graph there might be several possible extended graphs and hence several possible *corner assignments.* In many cases we assume that a corner assignment, and hence an extended graph, has already been fixed, but if this is not the case then it is possible to test all corner assignments in polynomial time.

A rectangular layout $\mathcal{L}$ naturally induces a labeling of its extended dual graph $E(\mathcal{G})$. If two rectangles of $\mathcal{L}$ share a vertical segment, then we color the corresponding edge in $E(\mathcal{G})$ blue (solid) and direct it from left to right. Correspondingly, if two rectangles of $\mathcal{L}$ share a horizontal segment, then we color the corresponding edge in $E(\mathcal{G})$ red (dashed) and direct it from bottom to top (Fig. 4). This labeling has the following properties: (i) around each inner vertex in clockwise order we have four contiguous sets of incoming blue edges, outgoing red edges, outgoing blue edges, and incoming red edges; (ii) the left exterior vertex has

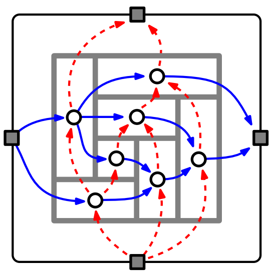


Figure 4: A rectangular layout and the regular edge labeling of its extended dual.

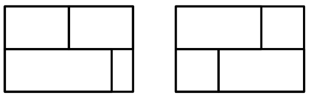


Figure 5: Two inequivalent but order-equivalent rectangular layouts.

only blue outgoing edges, the top exterior vertex has only red incoming edges, the right exterior vertex has only blue incoming edges, and the bottom exterior vertex has only red outgoing edges.

Such a labeling is called a *regular edge labeling*. It was introduced by Kant and He [5] who showed that every regular edge labeling of an extended graph $E(\mathcal{G})$ uniquely defines an equivalence class of rectangular duals of a proper graph $\mathcal{G}$. Given any extended graph $E(\mathcal{G})$, a regular edge labeling for $E(\mathcal{G})$ can be found in linear time and the rectangular dual defined by it can also be constructed in linear time [5].

Two layouts $\mathcal{L}$ and $\mathcal{L}'$ are *equivalent*, denoted by $\mathcal{L} \sim \mathcal{L}'$, if they induce the same regular edge labeling of the same dual graph. We say that a rectangular layout $\mathcal{L}$ with n rectangles $R_1, ..., R_n$ realizes a weight function $w : R_1, ..., R_n \to \mathbb{R}, w(i) > 0$ as a *rectangular cartogram* if there exists a layout $\mathcal{L}' \sim \mathcal{L}$ such that for any $1 \leq i \leq n$ the area of rectangle R_i equals $w(r_i)$. Correspondingly, we say that a layout $\mathcal{L}$ realizes w as a *perimeter cartogram* if there exists a layout $\mathcal{L}' \sim \mathcal{L}$ such that the perimeter of each rectangle of $\mathcal{L}'$ equals the prescribed weight. A layout $\mathcal{L}$ is *area-universal* if it realizes every possible weight function.

It is convenient to define a weaker equivalence relation on layouts than equivalence, which we call *order-equivalence*. For a layout $\mathcal{L}$, we define a partial order on the vertical maximal segments, in which $s_1 \leq s_2$ if there exists an x-monotone curve that has its left endpoint on s_1, its right endpoint on s_2, and that does not cross any horizontal maximal segments. We define a partial order on the horizontal segments in a symmetric way. $\mathcal{L}$ and $\mathcal{L}'$ are order-equivalent if their rectangles and maximal segments correspond one-for-one in a way that preserves these partial orders.

Observation 1 *A rectangular layout with n rectangular regions has $n - 1$ maximal segments.*

3 There can be only one

We first show that for any combination of layout and weight function there can be at most one rectangular cartogram or perimeter cartogram. More generally, if two geometrically different but order-equivalent layouts share the same bounding box, there is a rectangle in one of the layouts that is larger in both of its dimensions than the corresponding rectangle in the other layout. Thus, let $\mathcal{L}$ and $\mathcal{L}'$ be two geometrically different order-equivalent layouts with the same bounding box. The *push graph* $\mathcal{H}$ of $\mathcal{L}$ and $\mathcal{L}'$ is a directed graph that has a vertex for each rectangle in $\mathcal{L}$ and an edge from vertex R_i to vertex R_j if the rectangles R_i and R_j are adjacent and the maximal segment in $\mathcal{L}$ that separates R_i from R_j is shifted in $\mathcal{L}'$ towards R_j and away from R_i.

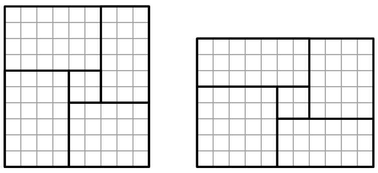


Figure 6: Two equivalent layouts in which corresponding rectangles have the same perimeter.

Lemma 1 *The push graph for $\mathcal{L}$ and $\mathcal{L}'$ contains a node with no incoming or no outgoing edges.*

Theorem 2 *For any layout $\mathcal{L}$ and any weight function w there is at most one layout $\mathcal{L}'$ (up to affine transformations) that is order-equivalent to $\mathcal{L}$ and that realizes w as a rectangular cartogram.*

For perimeter, such strong uniqueness does not hold: there are equivalent layouts that are not affine transformations of each other in which the perimeters of corresponding rectangles are equal (Fig. 6). However, if we fix the outer bounding box of the layout, the same proof method works:

Theorem 3 *For any layout $\mathcal{L}$ and any weight function w there is at most one layout $\mathcal{L}'$ that is order-equivalent to $\mathcal{L}$ with the same bounding box and that realizes w as a perimeter cartogram.*

More generally the same result holds for any type of cartogram in which rectangle sizes are measured by any strictly monotonic function of the height and width of the rectangles.

4 Area-universality and one-sidedness

All layouts are area-universal in a weak sense involving order-equivalence in place of equivalence. The proof of Lemma 4 uses Theorem 2 to invert the map W from vectors of positions of segments in a layout to vectors of rectangle areas, along a line segment from the area vector of $\mathcal{L}$ to the desired area vector.

Lemma 4 *For any layout $\mathcal{L}$ and weight function w, there exists a layout $\mathcal{L}'$ that has a square outer rectangle, is order-equivalent to $\mathcal{L}$, and realizes w as a rectangular cartogram.*

One may find $\mathcal{L}'$ by hill-climbing to reduce the Euclidean distance between the current weight function and the desired weight function. No layout $\mathcal{L}$ can be locally but not globally optimal, because within any neighborhood of $\mathcal{L}$ the inverse image of the line segment connecting its weight vector to the desired weight vector contains layouts that are closer to w. Alternatively, one can find $\mathcal{L}'$ by a numerical procedure that follows this inverse image by inverting the Jacobean matrix of W at each step. We do not know whether it is always possible to find $\mathcal{L}'$ exactly by an efficient combinatorial algorithm (as may easily be done for the subclass of sliceable layouts), or whether the general solution involves roots of high-degree polynomials that can be found only numerically.

Theorem 5 *The following three properties of a layout $\mathcal{L}$ are equivalent:*

1. *$\mathcal{L}$ is area-universal.*
2. *Every layout that is order-equivalent to $\mathcal{L}$ is equivalent to $\mathcal{L}$.*
3. *$\mathcal{L}$ is one-sided.*

5 Finding perimeter cartograms

Although our proof of uniqueness for rectangular cartograms generalizes to perimeter, our proof that any layout and weight function have a realization as an order-equivalent cartogram does not generalize: there exist one-sided layouts and weight functions that cannot be realized as a perimeter cartogram (Fig. 7).

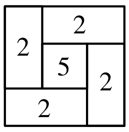


Figure 7: The outer rectangles contribute at most one unit of shared boundary to the perimeter of the central rectangle, which is too large to be realized.

Nevertheless, one can test in polynomial time whether a solution exists for any layout and weight function. The technique involves describing the constraints on the perimeters of rectangles as linear equalities that reduce the dimension of the space of layouts to at most two, and forming a low-dimensional linear program from inequality constraints expressing the equivalence to $\mathcal{L}$ of the other layouts within this low-dimensional space.

Theorem 6 *For any layout $\mathcal{L}$ and any weight function w we can find a layout $\mathcal{L}'$ that is equivalent to $\mathcal{L}$ and that realizes w as a perimeter cartogram, if one exists.*

The same algorithm can be used to find an order-equivalent layout rather than an equivalent layout, by restricting the inequality constraints to the subset that determine order-equivalence.

6 Finding one-sided layouts

Recall that every proper triangulated plane graph has a rectangular dual, but not necessarily a one-sided rectangular dual. Since one-sided duals are area-universal, it is of interest to find a one-sided dual for a proper graph if one exists. Our overall approach is, first, to partition the graph on its separating four-cycles; second, to represent the family of all layouts for a proper graph as a distributive lattice, following Fusy [4]; third, to represent elements of the distributive lattice as partitions of a partial order according to Birkhoff's theorem [1]; fourth, to characterize the ordered partitions that correspond to one-sided layouts; and fifth, to search in the partial order for partitions of this type. This approach does not yield polynomial time algorithms, but they are polynomial whenever the number of separating four-cycles in the given proper graph is bounded by a fixed constant, or more generally when such a bound can be given separately within each of the pieces found in the partition we find in the first stage of our algorithms.

References

[1] G. Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.

[2] D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek. Area-Universal Rectangular Layouts. `http://arxiv.org/abs/0901.3924v1`, 2009.

[3] C. F. Earl and L. J. March. Architectural applications of graph theory. In R. Wilson and L. Beineke, editors, *Applications of Graph Theory*, pp. 327–355. Academic Press, London, 1979.

[4] É. Fusy. Transversal structures on triangulations: A combinatorial study and straight-line drawings. *Discrete Mathematics*, 2009. To appear.

[5] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172(1–2):175–193, 1997.

[6] K. Koźmiński and E. Kinnen. Rectangular duals of planar graphs. *Networks*, 5(2):145–157, 1985.

[7] E. Raisz. The rectangular statistical cartogram. *Geographical Review*, 24(2):292–296, 1934.

[8] I. Rinsma. Nonexistence of a certain rectangular floorplan with specified areas and adjacency. *Environment and Planning B: Planning and Design*, 14(2):163–166, 1987.

[9] M. van Kreveld and B. Speckmann. On rectangular cartograms. *Comp. Geom.: Theory and Applications*, 37(3):175–187, 2007.

[10] G. K. H. Yeap and M. Sarrafzadeh. Sliceable floorplanning by graph dualization. *SIAM J. Discrete Mathematics*, 8(2):258–280, 1995.

Constructability of Trip-lets

Jeroen Keiren[*] Freek van Walderveen[†] Alexander Wolff[*]

Abstract

A trip-let is an object as shown on the cover of Hofstadter's book *Gödel, Escher, Bach*: a solid, three-dimensional object that, when viewed from three orthogonal directions, shows three different letters. In this paper we consider two problems related to the construction of such objects for a given set of three letters. First, we want to know whether the silhouettes of the object correspond to the letters we used to make the object. Second, we are interested in the connectedness of the final object: does it fall apart during construction? We obtain results on the combinatorial complexity of objects and silhouettes for letters given as general or rectilinear polygons with holes, and give algorithms to solve the problems efficiently for the rectilinear case.

1 Introduction

The process of making a trip-let can be described mathematically as intersecting a set of infinite cones, where each cone has its apex at the position of the viewer and the shape as its base. In this paper, we only consider orthogonal projections, thus cones become infinitely long prisms.

Definition 1 *The z-*prism *induced by a polygonal shape (with holes) $S_{xy} \subset \mathbb{R}^2$ is the volume*

$$P_z(S_{xy}) := \{(x, y, z) \mid (x, y) \in S_{xy}, z \in \mathbb{R}\} \subset \mathbb{R}^3.$$

We define $P_y(S_{xz})$ and $P_x(S_{yz})$ analogously.

Definition 2 *The* trip-let *obtained from three given shapes S_{xy}, S_{xz}, and S_{yz} is the intersection of their prisms: $P_z(S_{xy}) \cap P_y(S_{xz}) \cap P_x(S_{yz}) \subset \mathbb{R}^3$.*

In this paper we consider two properties of trip-lets; *validity* and *connectedness*. A connected trip-let is simply a trip-let consisting of one solid, connected piece. To define the validity of a trip-let, we need the notion of silhouette.

Definition 3 *The z-*silhouette *of a volume V is*

$$\pi_{xy}(V) := \{(x, y) \mid \exists z \in \mathbb{R} : (x, y, z) \in V\}.$$

Again, we define x- and y-silhouette similarly.

We say that a trip-let is valid if each of its silhouettes matches exactly the corresponding input shape S_{xy}, S_{xz}, and S_{yz}. The example shown in Figure 1(a) illustrates this. Although the intersection of any combination of two of the polygons has correct silhouettes, adding the third polygon yields an invalid triplet. Hence, any algorithm solving this problem has to consider all three shapes together. Figure 1(b) illustrates the connectedness problem: in the centre of the trip-let there is a small cube that is not connected to the rest of the object.

At the 1998 *ACM Symposium on Computational Geometry*, O'Rourke [1] presented the problem of finding "simple conditions" to determine whether three letters can form a valid and connected trip-let. We present a first step into this direction. We analyze the combinatorial complexity of triplets and silhouettes, and we give algorithms for testing the validity and connectedness of triplets induced by *rectilinear* polygons, that is, polygons whose edges are parallel to one of the primary axes. We measure the complexity of our algorithms with respect to the total number n

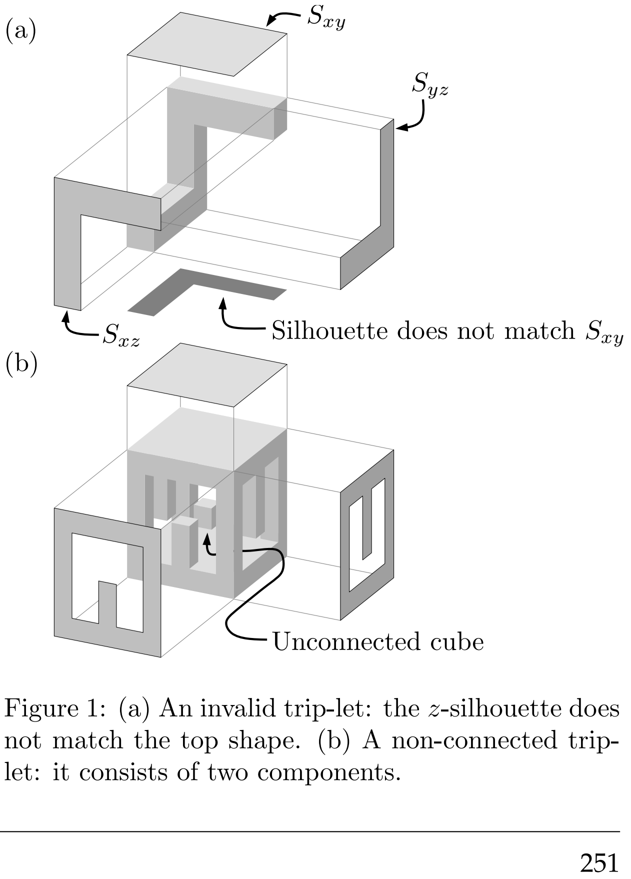


Figure 1: (a) An invalid trip-let: the z-silhouette does not match the top shape. (b) A non-connected triplet: it consists of two components.

[*]Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, The Netherlands. `j.j.a.keiren@student.tue.nl`, `www.win.tue.nl/~awolff`

[†]MADALGO, Department of Computer Science, University of Aarhus, Denmark. `freek@vanwal.dk`

of vertices of the input polygons.

Apart from their nice visual appearance, understanding the class of objects that can be represented by trip-lets may yield useful insights in our ability to understand three-dimensional scenes from silhouettes obtained from different viewpoints [7].

Earlier work Most work related to the construction of 3D objects from shapes considers the reconstruction of actual objects from silhouettes obtained from different viewpoints. Our orthogonal case can be seen as a variant of this problem with viewpoints at infinity where we are not sure whether the object to be reconstructed actually exists.

Bottino and Laurentini [4] ask whether it is possible to find the relative positions of viewpoints parallel to a plane, given their corresponding silhouettes. If such positions can be found, they call the silhouettes *compatible*. They give sets of inequalities for the relative positions of the viewpoints for viewing directions parallel to the plane, but do not provide efficient algorithms to find these positions.

Ohgami and Sugihara [8] propose an algorithm for finding relative viewing points in the plane for three given silhouettes. Their algorithm does not guarantee to find feasible positions even if they exist.

Hachenberger and Kettner [5] describe the implementation of Boolean operations on 3D Nef polyhedra in CGAL. As a prism can be described by a 3D Nef polyhedron, trip-lets can be constructed solely using Boolean operations on Nef polyhedra. Then, the number of components of the resulting trip-let is known, solving our first problem. The silhouettes can be found by projecting the resulting polyhedron back to planes orthogonal to any of the three axes. By comparing the silhouettes with the input shapes, our second problem is also solved.

The expected running time of the intersection implementation described by Hachenberger and Kettner is dominated by the term $O(k\sqrt[3]{k}\log k)$, where k is the size of the resulting polyhedron. Hence, the above solution runs in $O(n^4 \log n)$ expected time if the object consists of $\Theta(n^3)$ cubes (see Fig. 2 for an example of such an object).

Our results First, we investigate (in Section 2) the combinatorial complexity of our objects: given three shapes of total complexity n, we show that the resulting trip-let and its silhouettes have complexity $\Theta(n^3)$ and $\Theta(n^4)$, respectively. Given rectilinear shapes, we show that in the worst case a silhouette has complexity $\Theta(n^2)$.

Second, we give algorithms for testing validity and connectedness of trip-lets constructed from rectilinear shapes, see Sections 3 and 4. Within the class of algorithms that actually construct the silhouettes and trip-lets, our algorithms are near-optimal in the worst case, testing validity and connectedness in time $O(n^2 \log n)$ and $O((n^2 + k)\log n)$, respectively. Note that k, the complexity of the given trip-let, is $\Omega(n^2) \cap O(n^3)$ in the worst case.

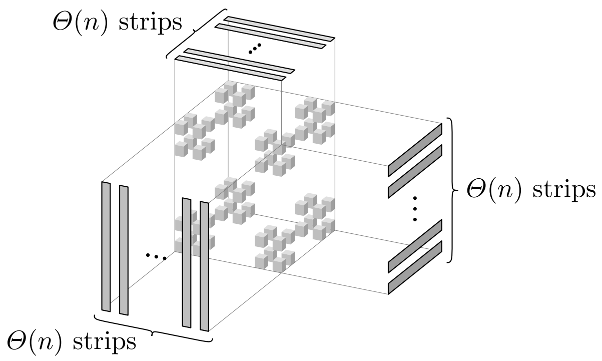


Figure 2: Three shapes generating $\Theta(n^3)$ cubes.

2 Complexity of objects and silhouettes

Determining the worst-case complexity of objects and silhouettes by bounding the respective number of vertices allows us to prove lower bounds on the running time of any algorithm that, in answering one of the problems, internally constructs the actual object or silhouette. Due to lack of space, we only state the results and give some of the more important examples.

We start with a lemma concerning the complexity of objects obtained from two or three shapes.

Lemma 1 *Given $m \in \{2, 3\}$ shapes with n vertices in total, the complexity of the intersection of their orthogonally oriented prisms is $\Theta(n^m)$ in the worst case.*

An example illustrating the lower bound is given in Figure 2. The set of three shapes yields an object consisting of $\Theta(n^3)$ vertices. Dropping one of the shapes results in an object with $\Theta(n^2)$ vertices.

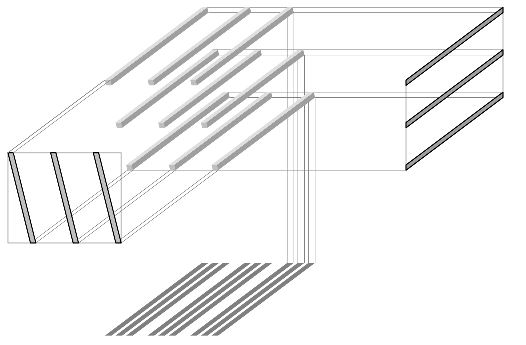
Note that the trip-let in Figure 2 is not connected. We conjecture the following.

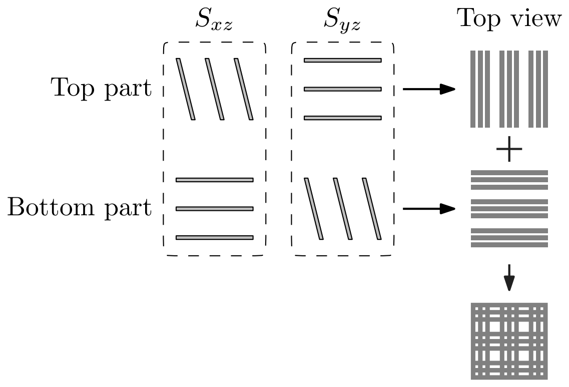
Conjecture 1 *Connected trip-lets have $O(n^2)$ vertices.*

It is easy to construct a connected trip-let with $\Theta(n^2)$ vertices.

The following lemma shows that silhouettes can have higher complexity if we allow general polygonal shapes.

Lemma 2 *A silhouette constructed from general polygonal shapes of total complexity n has complexity $\Theta(n^4)$ in the worst case.*


(a)



(b)

Figure 3: (a) Construction with $3 \times 3 = 9$ bars that do not overlap in the z-silhouette. (b) Construction with $(9-1) \times (9-1) = 64$ holes in the z-silhouette.

For proving the lower bound, consider one shape with $\Theta(n)$ parallel strips that are slanted but do not overlap when projected vertically, and one with $\Theta(n)$ parallel, horizontal strips such as in Figure 3(a). The z-silhouette obtained from these two shapes consists of $\Theta(n^2)$ parallel strips. By swapping the two shapes the silhouette turns 90 degrees. If we stack the original and rotated version, the combined silhouette is a grid pattern with $\Theta(n^4)$ vertices (see Figure 3(b)).

Hence, any algorithm that internally constructs the silhouettes to be verified can never be faster than $\Omega(n^4)$. Fortunately, if we restrict ourselves to rectilinear shapes, things cannot get that bad.

Lemma 3 *A silhouette constructed from rectilinear shapes of total complexity n has complexity $\Theta(n^2)$ in the worst case.*

3 Determining validity

The correctness of one of the silhouettes of a trip-let, say the one for shape S_{xy}, can be stated by saying that the intersection of the prisms induced by the other shapes "overshadows" S_{xy}:

$$S_{xy} \subseteq \pi_{xy}(P_y(S_{xz}) \cap P_x(S_{yz})).$$

We can use this characterization as a general idea for an algorithm to determine the validity of a trip-let. If the silhouette of the intersection of two prisms is known, a simple line-sweep suffices to determine if this silhouette covers the third shape. As we saw in Section 2, general polygonal shapes may yield silhouettes of complexity $\Theta(n^4)$. We expect we cannot easily get better running times than we saw for the approach outlined in the introduction (at least for the validity problem). Therefore, from now on, we will only consider shapes described by rectilinear polygons.

We compute the orthogonal projection of the intersection of the prisms formed by two silhouettes using a

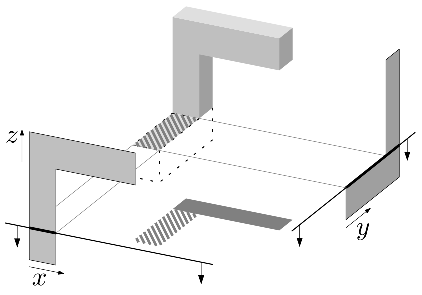


Figure 4: Constructing the bottom silhouette; the dashed area is a new rectangular facet.

sweep-plane algorithm. The orientation of the sweep plane (effectively consisting of two sweep lines) is illustrated in Figure 4. For each event—that is, a horizontal edge—the new facets of the trip-let introduced by that event that are parallel to the sweep plane are computed. This results in a set of rectangles whose union is exactly the orthogonal projection we were looking for. As this is a fairly standard procedure, we will not go into further detail.

The main problem with actually computing the union of this set of rectangles is that there may for example be $\Theta(n^2)$ rectangles whose union again has a complexity of $\Theta(n^2)$ (consider for example a rectilinear version of Figure 3), so adding one rectangle at a time will yield a running time of $\Theta(n^3)$ or worse. Instead, we determine the union in one go using a sweep-line algorithm. We maintain the status of the sweep line in a data structure that can efficiently (i) add a segment when a new rectangle is encountered, (ii) report the intervals covered by this new segment that are not currently covered (as they introduce new edges in the silhouette polygon), and (iii) remove a segment when the sweep line hits the bottom of a rectangle. Segment trees [3, Section 10.3] come close but do not provide an efficient implementation of the second operation: reporting k empty intervals within a given range in $O((k+1)\log n)$ time. We solve this by augmenting the segment tree with a field for every node ν that indicates whether the interval corresponding to ν is completely covered by segments that end somewhere inside this interval. Combining this with information about the segments completely covering this interval that is already present in the segment tree, we can implement the empty-interval query in the given time complexity.

Theorem 4 *The validity of a trip-let obtained from three rectilinear shapes of total complexity n can be determined in $O(n^2 \log n)$ time.*

A formal proof is omitted due to lack of space.

4 Determining connectedness

To check whether the intersection of three orthogonal prisms is connected, we again use a sweep-plane algorithm. In short, it maintains the intersection of the sweep plane with the object, which we call a *section*, and keeps a data structure representing the connectedness of the components of the object already passed by the sweep plane. Each time a new part of the object hits the sweep plane, the algorithm determines whether this part is connected to any other known part and perhaps even connects two parts that were not connected before. Then, using a union-find data structure, we can determine whether the final object is connected. Our objective is to make the running time of the algorithm linearly dependent on the complexity of the object, up to a polylogarithmic factor.

Because our algorithm effectively calculates the intersection of the three prisms, it may be compared to the algorithm for Boolean operations on orthogonal polyhedra by Aguilera and Ayala [2][1]. The key difference with this algorithm is in the way new sections are computed. Instead of doing the same Boolean operation (intersection in our case) on two objects of lower dimension (the rectilinear polygon forming the intersection of the sweep plane and two of the three prism on the one hand, and the third shape on the other hand), we update our frame incrementally so that we do not spend time on parts of a section that will not cause new vertices of the final object.

This way, we can determine whether a trip-let from rectilinear shapes is connected in $O((n^2+k)\log n)$ time. Combining this result with that from the previous section, we obtain our last theorem.

Theorem 5 *Given three rectilinear shapes with n vertices in total, we can determine whether the intersection of the prisms induced by these shapes forms a valid and connected trip-let in $O((n^2+k)\log n)$ time, where k is the complexity of the trip-let.*

[1]We do not understand Aguilera and Ayala's claim that their algorithm runs in time linear in the number of vertices of the input polyhedra.

Note that by Lemma 1, $k = O(n^3)$ in the worst case; if Conjecture 1 holds, the algorithms run in $O(n^2 \log n)$ time.

5 Open problems

The running times of our algorithms for rectilinear input shapes are almost optimal within the class of algorithms that actually construct the objects or silhouettes. Finding the conditions O'Rourke is looking for, or the problem of whether it is possible to do better by somehow finding the answers without constructing the silhouettes or objects first, is still open: we do not know of any non-trivial lower bounds for this case yet. Furthermore, does our conjecture hold, that is, does any connected trip-let have at most quadratic complexity?

Acknowledgments

We thank Christ van Willegen for bringing these problems to our attention.

References

[1] P. K. Agarwal and J. O'Rourke. Computational Geometry Column 34. *SIGACT News* 29(3):27–32, 1998. `http://www.cs.duke.edu/~pankaj/scg98-openprobs/`.

[2] A. Aguilera and D. Ayala. Orthogonal polyhedra as geometric bounds in constructive solid geometry. *Proc. 4th ACM Symp. Solid Modeling and Applications*, pp. 56–67, 1997.

[3] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, third edition, 2008.

[4] A. Bottino and A. Laurentini. Introducing a new problem: Shape-from-silhouette when the relative positions of the viewpoints is unknown. *IEEE Trans. Pattern Analysis and Machine Intel.*, 25(11):1484–1493, 2003.

[5] P. Hachenberger, L. Kettner, and K. Mehlhorn. Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. *Comput. Geom. Theory Appl.*, 38(1–2):64–99, 2007.

[6] D. R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books, 1979.

[7] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Trans. Pattern Analysis and Machine Intel.*, 16(2):150–162, 1994.

[8] T. Ohgami and K. Sugihara. Realizability of solids from three silhouettes. In *Abstracts 24th European Workshop Comput. Geom.*, pp. 233–236, 2008.

Finding Perfect Auto-Partitions is NP-hard[*]

Mark de Berg[†] Amirali Khosravi[†]

Abstract

A perfect BSP for a set S of disjoint line segments in the plane is a BSP in which none of the objects is cut. We study a specific class of BSPs, called *auto-partitions* and we prove that it is NP-hard to find if a perfect auto-partition exists for a set of lines.

1 Introduction

Many problems involving objects in the plane or higher-dimensional space are solved more efficiently if a hierarchical partitioning of the space is given. One of the most popular hierarchical partitioning schemes is the *binary space partition*, or BSP for short. In a BSP the space is recursively partitioned by hyperplanes until there is at most one object intersecting the interior of each cell in the final partitioning. Note that the splitting hyperplanes not only partition the space, they may also cut the objects into fragments.

The recursive partitioning can be modeled by a tree structure, called a BSP *tree.* Nodes in a BSP tree corresponds to subspaces of the original space, with the root node corresponding to the whole space and the leaves corresponding to the cells in the final partitioning. Each internal node stores the hyperplane used to split the corresponding subspace, and each leaf stores the object fragment intersecting the corresponding cell.

BSPs have been used in numerous applications. In most of these applications, the efficiency is determined by the *size* of the BSP tree, which is equivalent to the total number of object fragments created by the partitioning process. As a result, many algorithms have been developed that create small BSPs. For example, Paterson and Yao [6] presented an algorithm that computes for any given set of n disjoint segments in the plane a BSP of size $O(n \log n)$. Also for many other settings—axis-parallel objects, 3-dimensional objects, fat objects, etc.—algorithms have been developed that produce provably small BSPs; for an overview of results and applications on BSPs see the survey paper by Tóth [7].

In all of the above algorithms, bounds are proved on the *worst-case size* of the computed BSP *over all sets of n input objects* from the class of objects being considered. For any particular input, one may be able to do much better than in a worst-case scenario. Ideally, one would like to have an algorithm that computes a BSP that is *optimal for the given input*, rather than optimal in the worst-case. For n axis-parallel segments in the plane one can compute an optimal rectilinear BSP in $O(n^5)$ time [2]. Another result related to optimal BSPs is that for any set of (not necessarily rectilinear) disjoint segments in the plane one can compute a *perfect* BSP in $O(n^2)$ time, if it exists [1]. (A perfect BSP is a BSP in which none of the objects is cut). If such a BSP does not exist, then the algorithm only reports this fact.

Thus, it is still unknown if it is possible to compute (or maybe approximate) an optimal BSP for a set of segments. We study this problem for a specific type of BSPs, called *auto-partitions.* Let S denote the set of n disjoint input segments, R the region which we want to partition at some point, and $S(R)$ the set of segment fragments in the interior of R. Then an auto-partition uses a splitting line that contains a segment from $S(R)$. Fig. 1 shows a general (unrestricted) BSP of a set of input segments and an auto-partition of it.

Since autopartitions are a restricted type of BSPs, our hope was that it would be easier to compute optimal autopartitions than it is to compute optimal unrestricted BSPs. Unfortunately, this turns out to be not the case: we show that even the problem of finding perfect autopartitions—that is, deciding if the minimum number of cuts is zero—is already NP-hard. This should be contrasted to the result mentioned above, that deciding whether a set of segments admits a perfect general (or unrestricted) BSP can be done in $O(n^2)$ time. Hence, optimal auto-partitions seem more difficult to compute than optimal unrestricted BSPs.

2 Hardness of computing perfect auto-partitions

We consider the following problem.

PERFECT AUTO-PARTITION
Input: A set S of n disjoint line segments in the plane.
Output: YES if S admits a perfect auto-partition(an auto-partition without cuts), NO otherwise.

We will show that PERFECT AUTO-PARTITION is

[*]This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

[†]Department of Computing Science, TU Eindhoven, mdberg@win.tue.nl, akhosrav@win.tue.nl

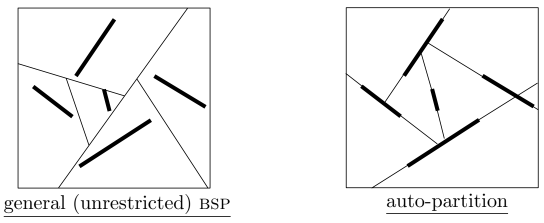


Figure 1: Two types of BSPs. Note that, as is usually done for auto-partitions, we have continued the auto-partition until the cells are empty.

NP-hard. Our proof is by reduction from a special version of the satisfiability problem, which we define and prove NP-complete in the next subsection. After that we prove the hardness of PERFECT AUTO-PARTITION.

2.1 Planar monotone 3-SAT

Let $\mathcal{U} := \{x_1, \dots, x_n\}$ be a set of n boolean variables, and let $\mathcal{C} := C_1 \wedge \cdots \wedge C_m$ be a CNF formula defined over these variables, where each clause C_i is the disjunction of at most three variables. 3-SAT is the problem of deciding whether such a boolean formula is satisfiable. An instance of 3-SAT is called *monotone* if each clause consists only of positive variables or only of negative variables. 3-SAT is NP-complete, even when restricted to monotone instances [3].

For a given (not necessarily monotone) 3-SAT instance, consider the bipartite graph $\mathcal{G} = (\mathcal{U} \cup \mathcal{C}, \mathcal{E})$, where there is an edge $(x_i, C_j) \in \mathcal{E}$ if and only if x_i or its negation $\overline{x_i}$ is one of the variables in the clause C_j. Liechtenstein [5] has shown that 3-SAT remains NP-complete when $\mathcal{G}$ is planar. Moreover, as shown by Knuth and Raghunatan [4], one can always draw the graph $\mathcal{G}$ of a planar 3-SAT instance such that the variables and clauses are drawn as rectangles with all the variable-rectangles on a horizontal line, the edges connecting the variables to the clauses are vertical segments, and the drawing is crossing-free. We call such a drawing of a planar 3-SAT instance a *rectilinear representation.* PLANAR 3-SAT remains NP-complete when a rectilinear representation is given.

Next we introduce a new version of 3-SAT, which combines the properties of monotone and planar instances. We call a clause with only positive variables a *positive clause*, a clause with only negative variables a *negative clause*, and a clause with both positive and negative variables a *mixed clause.* Thus a monotone 3-SAT instance does not have mixed clauses. Now consider a 3-SAT instance that is both planar and monotone. A *monotone rectilinear representation* of such an instance is a rectilinear representation where all positive clauses are drawn on the positive side of (that is, above) the variables and all negative clauses are drawn on the negative side of (that is, below) the variables. Our 3-SAT variant is defined as follows.

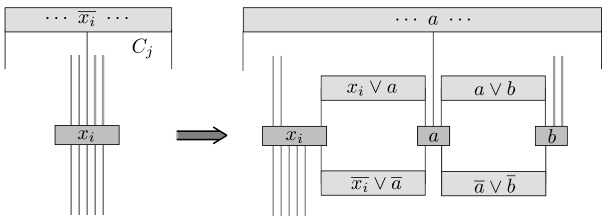

$\cdots\ \overline{x_i}\ \cdots$
C_j
x_i
$\cdots\ a\ \cdots$
$x_i \vee a$
$a \vee b$
x_i
a
b
$\overline{x_i} \vee \overline{a}$
$\overline{a} \vee \overline{b}$

Figure 2: replacing an inconsistent variable-clause.

PLANAR MONOTONE 3-SAT
Input: A monotone rectilinear representation of a planar monotone 3-SAT instance.
Output: YES if it is satisfiable, NO otherwise.

PLANAR MONOTONE 3-SAT is obviously in NP. We will prove that PLANAR MONOTONE 3-SAT is NP-hard by a reduction from PLANAR 3-SAT.

Let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a given rectilinear representation of a planar 3-SAT instance defined over the variable set $\mathcal{U} = \{x_1, \dots, x_n\}$. We call a variable-clause pair *inconsistent* if the variable is negative in that clause while the clause is placed on the positive side of the variables, or the variable is positive in the clause while the clause is placed on the negative side. If a rectilinear representation does not have inconsistent variable-clause pairs, then it must be monotone. Indeed, any monotone clause must be placed on the correct side of the variables, and there cannot be any mixed clauses because any mixed clause must form an inconsistent pair with at least one of its variables. We convert the given instance $\mathcal{C}$ step by step into an equivalent instance with a monotone planar representation, in each step reducing the number of inconsistent variable-clause pairs by one.

Let $(\overline{x_i}, C_j)$ be an inconsistent pair; inconsistent pairs involving a positive variable in a clause on the negative side can be handled similarly. We get rid of this inconsistent pair as follows. We introduce two new variables, a and b, and modify the set of clauses.

- In clause C_j, replace $\overline{x_i}$ by a.
- Introduce the following four clauses: $(x_i \vee a) \wedge (\overline{x_i} \vee \overline{a}) \wedge (a \vee b) \wedge (\overline{a} \vee \overline{b})$.
- In each clause containing x_i that is placed on the positive side of the variables and that connects to x_i to the right of C_j, replace x_i by b.

Let $\mathcal{C}'$ be the new set of clauses. The proof of the following lemma is omitted in this abstract.

Lemma 1 *$\mathcal{C}$ is satisfiable iff $\mathcal{C}'$ is satisfiable.*

Fig. 2 shows how this modification is reflected in the rectilinear representation.

By applying the above conversion to each of the at most $3m$ inconsistent variable-clause pairs, we obtain a 3-SAT instance with at most $13m$ clauses defined over at most $n + 6m$ variables. This new instance is satisfiable iff $\mathcal{C}$ is satisfiable, and it has a monotone representation. We get the following theorem.

Theorem 2 PLANAR MONOTONE 3-SAT *is* NP-*complete.*

2.2 From planar monotone 3-SAT to perfect auto-partitions

Let $\mathcal{C} = C_1 \wedge \cdots \wedge C_m$ be a planar monotone 3-SAT instance defined over a set $\mathcal{U} = \{x_1, \ldots, x_n\}$ of variables, with a monotone rectilinear representation. We will show how to construct a set S of line segments that admits a perfect auto-partition iff $\mathcal{C}$ is satisfiable. The idea behind the reduction is shown in Fig. 3.

The variable gadget. For each variable x_i there is a gadget consisting of two segments, s_i and $\overline{s_i}$. Setting $x_i =$ TRUE corresponds to extending s_i before $\overline{s_i}$, and setting $x_i =$ FALSE to extending $\overline{s_i}$ before s_i.

The clause gadget. For each clause C_j there is a gadget consisting of four segments, $t_{j,0}, \ldots, t_{j,3}$. The segments in a clause form a cycle, that is, the splitting line $\ell(t_{j,k})$ cuts the segment $t_{j,(k+1) \bmod 4}$. This means that a clause gadget in isolation, would generate at least one cut. Now suppose that the gadget for C_j is crossed by the splitting line $\ell(s_i)$ through the segment s_i in such a way that $\ell(s_i)$ separates the segments $t_{j,0}, t_{j,3}$ from $t_{j,1}, t_{j,2}$, as in Fig. 3. Then the cycle is broken by $\ell(s_i)$ and no cut is needed. This does not work when $\ell(\overline{s_i})$ is used before $\ell(s_i)$, since then $\ell(s_i)$ is blocked by $\ell(\overline{s_i})$ before crossing C_j.

The idea is as follows. For each clause $(x_i \vee x_j \vee x_k)$, we want to make the splitting lines $\ell(s_i)$, $\ell(s_j)$, and $\ell(s_k)$ all cross the clause gadget. Then by setting one of these variables to TRUE, the cycle is broken and no cuts are needed for the clause. We must be careful that the splitting lines are not blocked in the wrong way—for example, it could be problematic if $\ell(\overline{s_k})$ would block $\ell(s_i)$—and also that clause gadgets are only intersected by the splitting lines corresponding to the variables in that clause. In the remainder of this section we show how to overcome these problems.

Detailed construction. From now on we assume that the variables are numbered based on the monotone rectilinear representation, with x_1 being the leftmost and x_n being the rightmost variable.

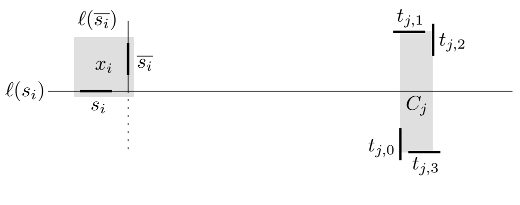


Figure 3: The idea behind the reduction.

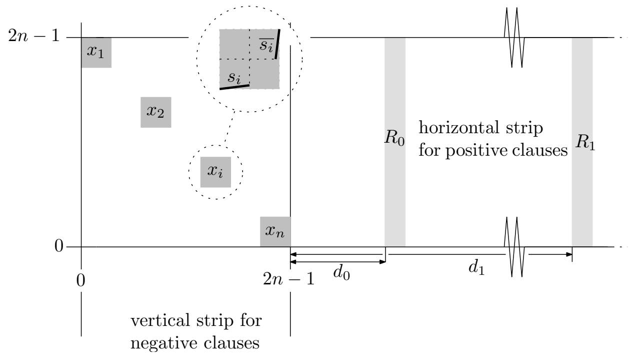


Figure 4: Placement of the variable gadgets and the clause gadgets (not to scale).

The gadget for a variable x_i will be placed inside the unit square $[2i-2 : 2i-1] \times [2n-2i : 2n-2i+1]$, as illustrated in Fig. 4. The segment s_i is placed with one endpoint at $(2i-2, 2n-2i)$ and the other endpoint at $(2i-\frac{3}{2}, 2n-2i+\varepsilon_i)$ for some $0 < \varepsilon_i < \frac{1}{4}$. The segment $\overline{s_i}$ is placed with one endpoint at $(2i-1, 2n-2i+1)$ and the other endpoint at $(2i-1-\overline{\varepsilon_i}, 2n-2i+\frac{1}{2})$ for some $0 < \overline{\varepsilon_i} < \frac{1}{4}$. Next we specify the slopes of the segments, which determine the values ε_i and $\overline{\varepsilon_i}$.

The gadgets for the positive clauses will be placed to the right of the variables, in the horizontal strip $[-\infty : \infty] \times [0 : 2n-1]$; the gadgets for the negative clauses will be placed below the variables, in the vertical strip $[0 : 2n-1] \times [-\infty : \infty]$. We describe how to choose the slopes of the segments s_i and how to place the positive clauses; the segments $\overline{s_i}$ and the negative clauses are handled in a similar fashion.

Consider the set $\mathcal{C}^+$ of all positive clauses in our 3-SAT instance, and the way they are placed in the monotone rectilinear representation. We call the clause directly enclosing a clause C_j the *parent* of C_j. Now let $\mathcal{G}^+ = (\mathcal{C}^+, \mathcal{E}^+)$ be the directed acyclic graph where each clause C_j has an edge to its parent (if it exists), and consider a topological order on the nodes of $\mathcal{G}^+$. We define the *rank* of a clause C_j, denoted by rank(C_j), to be its rank in this topological order. If rank$(C_j) = k$ then C_j is placed in a $1 \times (2n+1)$ rectangle R_k at distance d_k from the line $x = 2n-1$ (see Fig. 4), where $d_k := 2 \cdot (2n)^{k+1}$.

Before describing how the clause gadgets are placed inside these rectangles, we define the slopes of the segments s_i. Define rank(x_i), the rank of a variable x_i (with respect to the positive clauses), as the maximum rank of any clause it participates in. Now the slope of s_i is $\frac{1}{2 \cdot d_k}$, where $k = $ rank(x_i). Recall that x_i is placed inside the unit square $[2i-2 : 2i-1] \times [2n-2i : 2n-2i+1]$. The proof of the following lemma is omitted in this version.

Lemma 3 *Let x_i be a variable, and $\ell(s_i)$ be the splitting line containing s_i.*

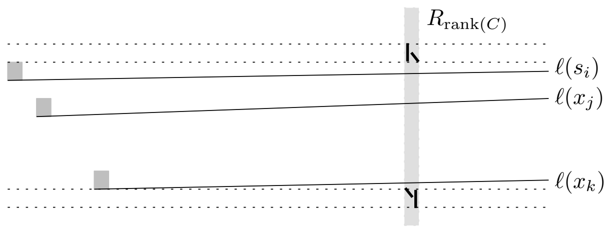


Figure 5: Placement of the segments forming a clause.

(i) For all x-coordinates in the interval $[2i-2 : 2n-1+d_{\text{rank}(x_i)}+1]$, $\ell(s_i)$ *has a y-coordinate in the range* $[2n-2i : 2n-2i+1]$.
(ii) $\ell(s_i)$ *intersects rectangles* $R_k (0 \leq k \leq \text{rank}(x_i))$.
(iii) $\ell(s_i)$ *does not intersect any rectangle* $R_k (k > \text{rank}(x_i))$.

We can now place the clause gadgets. Consider a clause $C = (x_i \vee x_j \vee x_k) \in \mathcal{C}^+$, with $i < j < k$; the case where C contains only two variables is similar. By Lemma 3(ii), the splitting lines $\ell(x_i), \ell(x_j), \ell(x_k)$ all intersect the rectangle $R_{\text{rank}(C)}$. Moreover, by Lemma 3(i) and since we have placed the variable gadgets one unit apart, there is a 1×1 square in $R_{\text{rank}(C)}$ just above $\ell(s_i)$ that is not intersected by any splitting line. Similarly, just below $\ell(s_k)$ there is a square that is not crossed. Hence, if we place the segments forming the clause gadget as in Fig. 5, then the segments will not be intersected by any splitting line. Moreover, the splitting lines of segments in the clause gadget—these segments either have slope -1 or are vertical—will not intersect any other clause gadget. This finishes the construction. The next lemma, whose proof is omitted, states the two key properties of construction. We say that, a splitting line $\ell(s_i)$ is *blocked* by $\ell(s_j)$ if $\ell(s_j)$ intersects $\ell(s_i)$ before $\ell(s_i)$ reaches $R_{\text{rank}(x_i)}$. This may prevent us from using $\ell(s_i)$ to resolve the cycle in the gadget of a clause containing x_i and is dangerous.

Lemma 4 *The variable and clause gadgets are placed such that the following holds:*
(i) The gadget for clause $(x_i \vee x_j \vee x_k)$ *is only intersected by the splitting lines* $\ell(s_i)$, $\ell(s_j)$ *and* $\ell(s_k)$. *Similarly, the gadget for clause* $(\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$ *is only intersected by the splitting lines* $\ell(\overline{s_i})$, $\ell(\overline{s_j})$ *and* $\ell(\overline{s_k})$.
(ii) A splitting line $\ell(s_i)$ *can only be blocked by a splitting line* $\ell(s_j)$ *or* $\ell(\overline{s_j})$ *when* $j \geq i$*; the same holds for* $\ell(\overline{s_i})$.

Theorem 5 PERFECT AUTO-PARTITION *is* NP-*complete.*

Proof. We can verify in polynomial time whether a given ordering of applying the splitting lines yields a perfect auto-partition, so PERFECT AUTO-PARTITION is in NP.

To prove that PERFECT AUTO-PARTITION is NP-hard, take a PLANAR MONOTONE 3-SAT instance and apply the above reduction to obtain a set S of $2n+4m$ segments forming an instance of PERFECT AUTO-PARTITION. The reduction can be done such that the segments have endpoints with integer coordinates of size $O(n^{2m})$, which means the number of bits for describing the instance is polynomial in $n+m$. It remains to show that $\mathcal{C}$ is satisfiable iff S has a perfect auto-partition.

Suppose S has a perfect auto-partition. Set $x_i :=$ TRUE if s_i is extended before $\overline{s_i}$, and $x_i :=$ FALSE otherwise. Since the auto-partition is perfect, the cycle in the gadget for a clause C must be broken. By Lemma 4(i) this can only be done by a line of one of the variables in the clause, say x_i. But then s_i has been extended before $\overline{s_i}(x_i = \text{TRUE})$ and C is true.

Now consider a truth assignment to the variables that satisfies $\mathcal{C}$. A perfect auto-partition for S can be obtained as follows. When $x_1 =$ TRUE we first take the splitting line $\ell(s_1)$ and then the splitting line $\ell(\overline{s_1})$; if $x_i =$ FALSE then we first take $\ell(\overline{s_1})$ and then $\ell(s_i)$. Next we treat s_2 and $\overline{s_2}$ in a similar way, and so on. So far we have not made any cuts. We claim that after having put all splitting lines $\ell(s_i)$ and $\ell(\overline{s_i})$ in this manner, we can put the splitting lines containing the segments in the clause gadgets, without making any cuts. Indeed, consider the gadget for a positive clause C. Because the truth assignment is satisfying, one of its variables, x_i, is TRUE. Then $\ell(s_i)$ is used before $\ell(\overline{s_i})$. Moreover, because we treated the segments in order, $\ell(s_i)$ is used before any other splitting lines $\ell(s_j)$, $\ell(\overline{s_j})$ with $j > i$ are used. By Lemma 4(ii) these are the only splitting lines that could block $\ell(s_i)$. Hence, $\ell(s_i)$ reaches the gadget for C and so we can resolve the cycle and get a perfect auto-partition. $\square$

References

[1] M. de Berg, M.M. de Groot, M.H. Overmars. Perfect binary space partitions. *Comput. Geom. Theory Appl.*,1997.

[2] M. de Berg, E. Mumford, B. Speckmann. Optimal BSPs and rectilinear cartograms. In *Proc. 14th Int. Symp. Advances Geographic Inf. Syst.*,2006.

[3] M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co.,1979.

[4] D.E. Knuth, A. Raghunathan. The problem of compatible representatives. *Discr. Comput. Math.*,1992.

[5] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*,1982.

[6] M.S. Paterson, F.F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discr. Comput. Geom.*,1990.

[7] C.D. Tóth, Binary space partitions: recent developments. *Combinat. and Comput. Geom.*, 2005.

Tight Spans and Coarsest Subdivisions of Convex Polytopes

Sven Herrmann*†

Abstract

Given a convex polytope P we study the coarsest polyhedral subdivisions of P. A main tool in this investigation is the tight span of a subdivision for which we show that it can be any polytope. The k-splits are defined as a special class of coarsest subdivisions and it is shown that they provide a suitable generalization of the splits, subdivisions into two parts. In particular, we show that k-splits are regular, leading to computable approximations for secondary polytopes.

1 Introduction

A *subdivision* of a convex polytope P is a collection of polytopes with union P such that all vertices of these polytopes are vertices of P and satisfying a certain intersection property. Subdivisions and especially triangulations (i.e., subdivisions into simplices) occur in various parts of mathematics; for an overview see the first chapter of the forthcoming book by de Loera, Rambau, and Santos [1]. It is an important structural result of Gel′fand, Kapranov, and Zelevinsky [2, Chapter 7] that there exists a polytope $\mathsf{SecPoly}(P)$ called the *secondary polytope* of P, whose vertices are the regular triangulations of P. Far less studied are the coarsest subdivisions of P (i.e., those subdivisions which cannot be refined non-trivially) who – if they are regular – correspond to the facets of $\mathsf{SecPoly}(P)$.

An object that can be used to study subdivisions of P is the *tight span*. This is a polyhedral complex dual to the complex of inner faces of a given subdivision. It can be used as a measure for the complexness of the subdivision. In Section 2, we will give the definition of the tight span and show that each polytope can occur as the tight span of some other polytope.

The simplest possible subdivisions of a polytope one can think of are those with exactly two maximal faces. These are called *splits* and were studied in detail by Joswig and the author [4] (see also Hirai [7]). Their tight spans are line segments. Results on splits were for example used in the context of the tropical Grassmannian (see [4, Section 7] and [3]). A polytope is called *totally splittable* if all its subdivisions can be obtained as common refinements of splits. A complete classification of these polytopes was given in [5]. The results of [4] and [5] are summarized in Section 3.

Motivated by these results, Section 4 serves as a starting point to the study of arbitrary coarsest subdivisions of P. We will see that such subdivisions can get quite complicated in general, hence, in Section 5, we propose a special class of coarsest subdivisions called k-splits, which share at least some properties of splits. In particular, we will prove that k-splits are regular subdivisions and hence facets of $\mathsf{SecPoly}(P)$. We conclude with some open questions. For the purpose of this extended abstract all proofs are omited.

2 Tight Spans

Let $P \subset \mathbb{R}^{d+1}$ be a polytope and $w : \mathsf{Vert}\, P \to \mathbb{R}$ a function assigning a weight to each vertex of P. If we lift each vertex according to its weight and project the lower faces of the resulting polytope down to P, we obtain a subdivision $\Sigma_w(P)$ of P. Subdivisions obtained in that way are called *regular*.

We now denote by V the matrix whose rows are the vertices of P and make the assumption that $P \subset \mathbb{R}^{d+1}$ is d-dimensional and that the vector $\mathbb{1} := (1, \ldots, 1)$ is contained in the linear span of the columns of V. The unbounded polyhedron

$$\mathcal{E}_w(P) \;:=\; \left\{ x \in \mathbb{R}^{d+1} \;\middle|\; Vx \geq -w \right\}$$

is called the *envelope* of P with respect to w. The complex of bounded faces of $\mathcal{E}_w(P)$ is the *tight span* $\mathcal{T}_w(P)$ of P with respect to w. The relation between regular subdivisions and tight spans is now the following.

Proposition 1 ([4], Prop. 2.3) *The face poset of $\mathcal{T}_w(P)$ is anti-isomorphic to the poset of interior faces of $\Sigma_w(P)$.*

This statement allows us to give a definition of the tight span also for non-regular subdivisions: For a subdivision Σ of P the (abstract) *tight span* $\mathcal{T}_\Sigma(P)$ of Σ is defined as the (abstract) polyhedral complex that is dual to the complex of inner faces of Σ. In this sense, for a weight function w the tight span $\mathcal{T}_w(P)$ is a realization of the abstract tight span $\mathcal{T}_{\Sigma_w(P)}(P)$.

One might ask which polyhedral complexes can occur as the tight span of a regular subdivision. Obviously, the complex has to be connected, and it follows from Hirai [6, Lemma 4.5] that the tight span of a regular subdivision is contractible and hence especially

*Fachbereich Mathematik, Technische Univeristät Darmstadt, Germany. sherrmann@mathematik.tu-darmstadt.de

†The author is supported by a Graduate Grant of TU Darmstadt.

simply connected. This can also be generalized to non-regular subdivisions. For the special case where the tight span consists of a sole maximal cell, the following theorem says that it can be any polytope.

Theorem 2 *Let P be any polytope. Then there exists a polytope P' and a lifting function w for P' such that $\mathcal{T}_w(P')$ is affinely isomorphic to P.*

3 Splits

A *split* S of a polytope P is a decomposition of P without new vertices which has exactly two maximal cells denoted by S_+ and S_-. By our technical assumptions on P, the linear span of $S_+ \cap S_-$ is a linear hyperplane H_S. Since S does not induce any new vertices, H_S does not meet any edge of P in its relative interior. Conversely, each hyperplane which separates P and which does not separate any edge defines a split of P. Furthermore, one sees that H defines a split if and only if it meets all faces (or facets) of P in a face of P or induces a split on them.

A decomposition of two weight function w_1, w_2 for a polytope P is called *coherent* if the subdivisions $\Sigma_{w_1}(P)$ and $\Sigma_{w_2}(P)$ have a common refinement. A weight function w is called *split prime* if $\Sigma_w(P)$ is not coarsened by any split. We can now formulate the main property of splits in the following Split Decomposition Theorem.

Theorem 3 ([7], Thm. 2.2; [4], Thm. 3.10) *Each weight function w has a coherent decomposition*

$$w = w_0 + \sum_{S \text{ split of } P} \lambda_S w_S ,$$

where w_0 is split prime, $\lambda_S \geq 0$, and this is unique among all coherent decompositions of w.

A polytope P is called *totally splittable* if all subdivisions of P are refinements of splits. Obviously, this is a severe restriction. However, we have the following complete characterization of these polytopes.

Theorem 4 ([5], Thm. 6) *A polytope is totally splittable if and only if it has the same oriented matroid as a simplex, a cross polytope, a polygon, a prism over a simplex, or a (possibly multiple) join of these polytopes.*

4 k-Subdivisions

For a polytope P we call a subdivision Σ of P a *k-subdivision* if it has k maximal faces and cannot be refined non-trivially. So splits are exactly the 2-subdivisions. By this definition, the set of all regular k-subdivisions (for all k) corresponds exactly to the set of facets of the secondary polytope of P.

For 2-subdivisions (i.e., splits) we know how the tight span looks like: it is a line segment. However, for k-subdivisions with $k \geq 3$ the tight spans get much more complicated. We will investigate this in this section. First, we give some general statements about the tight spans of k-subdivisions.

Proposition 5 *Let P be a polytope and Σ a k-subdivision of P with $k > 3$. Then the tight span $\mathcal{T}_\Sigma(P)$ is not a k-gon.*

Note that this only shows that k-gons with $k > 3$ cannot be the sole maximal cell of a tight span of a k-subdivision. It can well be that a polygon occurs as a maximal cell of a tight span of a k-subdivision if there are other maximal cells. For the most simple example see Figure 1; the tight span is depicted in red.

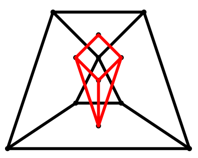


Figure 1: A quadrangle as maximal cell of the tight span of a 5-subdivisions.

Remark 1 *Almost all results of this paper can be generalized from polytopes to non-convex point configurations. One can also show that any tight span occurring for a point configuration occurs for some polytope (and for a coarsest subdivision of a point configuration if it is the tight span of a coarsest subdivision of a polytope). We use this to give lower-dimensional examples as in Figure 1.*

For the next condition we call a polyhedral complex 1-connected if it is still connected if one removes any vertex (i.e., the graph of the complex is 1-connected in the usual sense).

Proposition 6 *Let P be a polytope and Σ a k-subdivision of P. Then $\mathcal{T}_\Sigma(P)$ is 1–connected.*

As a further condition for the tight span of a k-subdivision, we recall from Section 2 that the tight span has to be simply connected. Additionally, this leads to the following important corollary.

Corollary 7 *Let P be a polytope, $k \geq 3$, and Σ a k-subdivision of P. Then all maximal faces of the polyhedral complex $\mathcal{T}_\Sigma(P)$ are at least two-dimensional.*

No we will examine the tight spans of k-subdivision for small k. If $k = 3$ we have to examine simply connected polyhedral complexes with three points. The only two possibilities are a triangle or two line segments connected at one point. The latter cannot occur by Proposition 7. This proves the following.

Lemma 8 *Let P be a polytope and Σ a 3-subdivision of P. Then the tight span of Σ is a triangle.*

For $k = 4$ we consider simply connected polyhedral complexes with four vertices. By Corollary 7, we have the additional condition that all maximal cells have to be at least two-dimensional. So the candidates are a tetrahedron, two triangles glued together at one edge, three triangles with a common vertex, or a quadrangle. Since the quadrangle cannot occur by Proposition 5, we have the following.

Lemma 9 *Let P be a polytope and Σ a 4-subdivision of P. Then the tight span of Σ is either a tetrahedron, consists of three triangles with a common vertex, or consists of two triangles glued together at one edge.*

For $k \geq 5$ the situation gets much more confusing. There are six possibilities for planar two-dimensional tight spans and five for pure three-dimensional tight spans. However, there exists also non-pure dimensional and non-planar two-dimensional tight spans. Additionally, we also have the first case where not all simply connected polyhedral complexes not excluded by Propositions 5 or 6 can occur as tight spans:

Lemma 10 *Let P be a polytope and Σ a 5–subdivision of P. Then the tight span of Σ cannot consist of a quadrangle and a triangle glued together at one edge.*

Lemma 10 can be extended to the statement that the tight span of any k-subdivision cannot be a $(k-1)$-gon glued with a triangle.

As we have seen in Lemmas 9 and 10, all three-dimensional polytopes with up to five vertices can appear as tight spans of k-subdivisions. Since all polytopes can occur as the tight span of some subdivision by Theorem 2, it seems natural to ask if all polytopes of dimension three or higher can occur as the tight span of some k-subdivision. The following proposition answers this question negatively.

Proposition 11 *Not all polytopes with dimension three or higher can occur as tight spans of k-subdivisions. Especially, a prism over a triangle cannot occur as the tight span of a 6-subdivision.*

5 k-Splits

As we have seen in the previous section, if k grows large there exist a lot of different possible tight spans for k-subdivisions. So an investigation of general k-subdivisions might be quite complicated. However, in this section, we will investigate for each k a special class of k-subdivisions which behave similar to splits: We call a k-subdivision Σ of P a *k-split* if Σ has an inner face of codimension $k-1$.

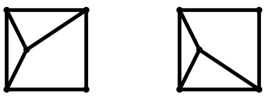


Figure 2: A point configuration with a codimension-two-face (the interior point) that corresponds to two different 3-splits.

It is easily seen that Σ is a k-split if and only if the tight span $\mathcal{T}_\Sigma(P)$ is a $(k-1)$-dimensional simplex. By Lemma 8, all k-subdivisions are k-splits for $k \leq 3$. Especially, the splits are the 2-splits.

Example 1 *As an example of a k-split, let P be the bipyramid over a $(k-1)$-dimensional simplex S. A k-subdivision Σ of P is obtained by taking as maximal faces of Σ the convex hulls of the edge connecting the two pyramid vertices and a facet of S.*

As to each split there corresponds a unique hyperplane, to each k-split there corresponds a unique subspace of codimension $k-1$. However, for splits we also have the property that if a hyperplane H defines a split, this split is uniquely determined by H. This does not hold any more for k-splits with $k \geq 3$; see Figure 2.

In Section 3, we saw that a hyperplane H defines a split of a polytope if and only if it does not meet any edge of P in its relative interior. One direction of this generalizes to k-splits as follows.

Proposition 12 *Let U be the unique codimension-$(k-1)$-subspace corresponding to some k-split of a polytope P. Then U meets all faces F of P in a face of P or corresponds to an m-split of F for some $m \leq k$. This condition is satisfied for all faces if its satisfied for all faces with $\dim F \leq k-1$ or for all facets.*

However, in contrast to the split case, the converse of Proposition 12 does not hold if $k \geq 3$. For an example, consider the polytope P depicted in Figure 3. The codimension-two-subspace spanned by the edge connecting the top and bottom vertices of P does not correspond to any 3-split.

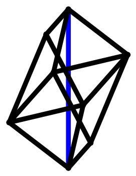


Figure 3: A polytope with an inner edge that does not correspond to a 3-split.

The most important property of splits is shared by k-splits: They are regular subdivisions.

Theorem 13 *All k-splits are regular.*

The proof of Theorem 13 explicitly constructs a weight function w inducing a given k-split, and so one gets an inequality defining the corresponding facet of the secondary polytope. This can be used to compute explicit (outer) approximations for $\mathsf{SecPoly}(P)$.

We call a polytope P *totally k-splittable* if all regular subdivisions of P are common refinements of m-splits with (possibly different) $m \leq k$. We call P *totally semi-splittable* if P is totally k-splittable for some k, or, equivalently, all regular subdivisions of P are common refinements of k-splits for some k. It is easily seen, that a d-dimensional polytope is totally semi-splittable if and only if it is totally d-splittable.

Example 2 *The 3-cube C_3 is totally 3-splittable, hence totally semi-splittable. It has 14 splits: Eight splits corresponds to cutting of one of of the vertices and six splits are defined by parallel pairs of diagonals in an opposite pair of cube facets. Furthermore, there are eight 3-splits: Each diagonal of the cube corresponds to two 3-splits by subdividing C_3 into three square pyramids with one of the vertices of the diagonal as apex. By using the facet-inequalities of $\mathsf{SecPoly}(C_3)$ for all those k-splits, we obtain a new computation of $\mathsf{SecPoly}(C_3)$, verifying the results of [9].*

Example 3 *The 4-cube C_4 is not totally semi-splittable. The secondary polytope of C_4 has 80,876 facets that come in 334 orbits (see [8]). Four of these orbits are splits, five are 3-splits, and three are 4-splits.*

One might ask whether there exists some generalization of the Split Decomposition Theorem 3 to k-splits. However, it can be easily seen that this is false even if one fixes some $k \geq 3$: The triangulation to the left of Figure 4 can be obtained as the common refinement of the 3-split A and either of the two 3-splits B_1, B_2 in Figure 4.

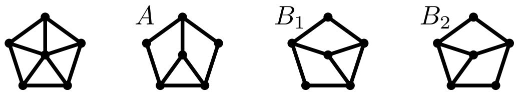


Figure 4: There is no unique 3-split decomposition.

6 Conclusions and Open Questions

We have discussed some conditions on when polyhedral complexes can be the tight span of some k-subdivision. However, we also gave examples that these conditions are not sufficient. For complexes with a sole maximal cell, we showed that the only possibility in dimension two is a triangle, and that in dimension three not all polytopes may occur. This naturally leads to the following question.

Question 1 *Which polyhedral complexes, especially, which polytopes occur as tight spans of k-subdivisions?*

It seams that k-splits are a very natural generalization of splits, so in view of Theorem 4 the following question arises.

Question 2 *Which polytopes are totally k-splittable or totally semi-splittable?*

The answer to this question might lead to interesting new classes of polytopes: the class of all totally 3-splittable polytopes, all totally 4-splittable polytopes, and so on. This would help to get new insights in the structure of secondary polytopes. Especially, since for the class of totally 2-splittable polytopes all secondary polytopes are known, a classification of totally k-splittable polytopes for small $k \geq 3$ could lead to explicit computations of some secondary polytopes.

References

[1] Jesús A. De Loera, Jörg Rambau, and Francisco Santos, *Triangulations: Structures and algorithms*, Springer, to appear.

[2] Israil M. Gel′fand, Mikhail M. Kapranov, and Andrey V. Zelevinsky, *Discriminants, resultants, and multidimensional determinants*, Mathematics: Theory & Applications, Birkhäuser Boston Inc., Boston, MA, 1994.

[3] Sven Herrmann, Anders N. Jensen, Michael Joswig, and Bernd Sturmfels, *How to draw tropical planes*, (2008), preprint `arXiv:0808.2383`.

[4] Sven Herrmann and Michael Joswig, *Splitting polytopes*, Münster J. Math. (2008), no. 1, 109–142.

[5] Sven Herrmann and Michael Joswig, *Totally splittable polytopes*, (2009), preprint `arXiv:0901.0231`.

[6] Hiroshi Hirai, *Characterization of the distance between subtrees of a tree by the associated tight span*, Ann. Comb. **10** (2006), no. 1, 111–128.

[7] Hiroshi Hirai, *A geometric study of the split decomposition*, Discrete Comput. Geome. **36** (2006), no. 2, 331–361.

[8] Peter Huggins, Bernd Sturmfels, Josephine Yu, and Debbie S. Yuster, *The hyperdeterminant and triangulations of the 4-cube*, Math. Comp. **77** (2008), 1653–1679.

[9] Julian Pfeifle, *The secondary polytope of the 3-cube*, Electronic Geometry Models (2000), `www.eg-models.de/2000.09.031`.

Practical Algorithms for Path Planning and Crowd Simulation

Mark H. Overmars[*]

Abstract

In computer games and other applications of virtual worlds many computer controlled characters move around. A fundamental challenge for these characters is to plan paths from their current locations to their desired locations. Such paths must not only be collision free, they must also be similar to the paths real people would take in such situations. This requires an adequate model of human navigation and generic algorithms that compute paths according to such a model. Such algorithms must be very fast as we usually can only use a fraction of the processing power and must compute paths for hundredths of characters in complicated environments.

In this talk I will give some insight in the criteria that must be satisfied by such practical path planning algorithms and I will describe the Indicative Route Method (IRM) that we developed to provide the required flexibility. This method computes networks of routes and collision-free corridors, based on Voronoi diagrams. For speed reasons we perform most computation on the GPU. The routes are only used as an indication of the desired paths, while the corridor provides local freedom for the characters to avoid each other and create natural motion. Also I will discuss new local character avoidance techniques that lead to considerably more natural simulations of large crowds than existing techniques.

[*]Institute of Information and Computing Sciences, Utrecht University, The Netherlands, `markov@cs.uu.nl`

Stabbers of line segments in the plane

M. Claverol[*] D. Garijo[†] C. I. Grima[‡] A. Márquez[§] C. Seara[¶]

Abstract

The problem of computing a representation of all the stabbing lines of a set of n line segments in the plane was solved by Edelsbrunner et al. with an optimal $\Theta(n \log n)$ time and $O(n)$ space algorithm. We present a complete study of different types of stabbers such as wedges, double-wedges, 3-polygonal chains and 2-level trees, providing algorithms whose complexities and spaces depend on the number of different slopes, and the number h_S of combinatorially different extreme lines or on the number c_S of combinatorially different critical extremes lines that appear in the set of segments. See Table 1.

1 Introduction

Let S be a set of n line segments in the plane. A line is said to be a *stabbing line* or a *transversal line* of S if it intersects each segment of the set S. Edelsbrunner et al. [3] presented an optimal $\Theta(n \log n)$ time and $O(n)$ space algorithm for computing and constructing a representation of all stabbing lines of S. However, the lower bound did not apply to the decision problem of determining whether there exists a stabbing line of S. Avis et al. [1] provided an $\Omega(n \log n)$ time lower bound in the fixed order algebraic decision tree model for studying the existence of a stabbing line.

Following this line of research, we address the problem of studying different types of simple stabbers, assuming that the segments of S are not stabbed by a line. Concretely, we shall consider the following structures (shown in Figure 1):

1. A *wedge* is the union of two rays with common origin, called the apex.

2. A *double-wedge* is formed by two lines intersecting at a point, called the vertex. As a particular case, we have two parallel lines.

3. A non-convex and simple 3-polygonal chain or *zig-zag* is composed by two rays and a segment joining the origin of both rays.

4. A 2-*level tree* is defined by one line and two rays located each on one of the half-planes defined by the line, and with origins on the line. In particular, we have the three parallel lines case.

Our main purpose is to design efficient algorithms for deciding whether there exists any of these structures stabbing the set S, and computing it in case of existence. As particular cases, we shall consider sets of parallel segments with or without the same length. A stabber can be viewed as a separator structure in

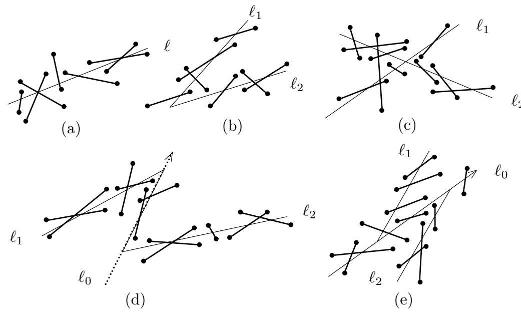


Figure 1: (a) Stabbing line, (b) stabbing wedge, (c) stabbing double-wedge, (d) stabbing zig-zag, (e) stabbing 2-level tree.

the sense that it can classify the endpoints of the segments, i.e., the structure splits the plane into disjoint monochromatic regions when the endpoints of the segments are assigned two different colors. Hurtado et al. [6] classified red and blue points in the plane by using similar separators to our stabbers. A stabber satisfy the *separability condition* when it is a separator structure. For instance, if a wedge W verify the separability condition then a segment of S can not be stabbed by the two rays of W. Otherwise, both rays can stab any segment of S.

Related works. Claverol [2] as a part of her PhD thesis initiated the study developed here in which we improve the complexities she obtained, and we also introduce some other stabbing problems. Edelsbrunner, Guibas and Sharir [4] showed how to construct a representation of line stabbers of convex polygons with a total of n vertices in $O(n\alpha(n) \log n)$ time. Later

[*]Dept. de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, Spain, merce@ma4.upc.edu
[†]Depto. de Matemática Aplicada I, Universidad de Sevilla, Sevilla, Spain, dgarijo@us.es
[‡]Depto. de Matemática Aplicada I, Universidad de Sevilla, Sevilla, Spain, grima@us.es
[§]Depto. de Matemática Aplicada I, Universidad de Sevilla, Sevilla, Spain, almar@us.es
[¶]Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain, carlos.seara@upc.edu

improved to $O(n \log n)$ using an $O(n \log n)$ time algorithm from Hershberger for finding the lower envelope of a segment set in the plane. O'Rourke [8] presented an algorithm for finding in case of existence a stabbing line of vertical line segments. Goodrich and Snoeyink [5] addressed a natural variant by solving the problem of computing a transversal convex polygon for a set of parallel segments in $O(n \log n)$ time. Lyons et al. [7] computed the minimum perimeter convex polygon which stabs a set of isothetic line segments. Rappaport [9] considered the problem of computing a simple polygon with minimum perimeter which stabs or contains a set of line segments.

2 Ideas and tools

Some ideas and tools are shared by most of our results. In this section we present them in a unified context. We first recall a standard geometric tool which is used throughout this paper (and in many other works on stabbers). This is the concept of *duality* [3], given by the geometric transform, denoted by $\mathcal{D}$, which maps a point into a non-vertical line and vice versa.

The lines containing the segments of S can have different slopes. Denote by m_i the slope of (the line containing) the segment $s_i \in S$. The complexity of many of the algorithms that we present here depends on the number of different slopes of the lines containing the segments of S, written as k_S. It is due to the following fact: the endpoints of the segments fall into two classes determined by a stabbing line ℓ, endpoints above ℓ and endpoints below ℓ, say red and blue respectively. The endpoint of a segment on ℓ is classified according to its other endpoint. Thus, for a given slope of ℓ our problem of stabbing the set S can be viewed as a red-blue separability problem of classifying the endpoints of the segments into disjoint monochromatic regions of the plane determined by the stabber. In fact, it is not difficult to show that there exist as many different classifications of the endpoints of S as k_S.

A relevant property about our stabbers is that not all the lines can be candidate to define one of these structures. For instance, consider a ray ℓ which is part of a wedge that stabs S, and its extension denoted by ℓ'. We have that all the segments that are not intersected by ℓ must lie on the same half-plane defined by ℓ'. Thus, we say that a line ℓ' is an *extreme line* for S if ℓ' stabs a subset of segments $S_1 \subseteq S$, $S_1 \neq \emptyset$, and the remaining segments $S_2 = S \setminus S_1$ are in only one of the open half-planes defined by ℓ'. Otherwise, the line ℓ' is said to be a non-extreme line for S. Thus, as we have mentioned before, our interest in extreme lines comes from the following fact: *the line containing any ray of a stabbing wedge for S is extreme line for S.*

In this paper, extreme lines are studied from two different points of view: computational and combinatorial view. The reason is that our algorithms depend on the computation of the set of extreme lines for S. Thus, we say that two non-vertical lines, ℓ_1 and ℓ_2, are *combinatorially different* with respect to S if either: (1) the subset of segments $S_1 \subseteq S$ stabbed by ℓ_1 is different from the subset of segments $S_2 \subseteq S$ stabbed by ℓ_2; or (2) if $S_1 = S_2$ then either: (i) the subset of endpoints of segments of S above ℓ_1 is different from the subset of endpoints of segments of S above ℓ_2, or (ii) the subset of endpoints of segments of S below ℓ_1 is different from the subset of endpoints of segments of S below ℓ_2. If h_S is the number of combinatorially different extreme lines of S, we compute a representation of the combinatorially different extreme lines for S in $O(h_S + n \log n)$ time and $O(h_S + n)$ space. Observe that the number of combinatorially different extreme lines for S is at most $O(n^2)$, but depending on the properties of the stabbing problem, we can consider only a subset of them named the *critical extreme lines* which size c_s is at most linear.

3 Results

3.1 Stabbing wedge

Our first aim is to study the problem of deciding whether the set S can be stabbed by a wedge, and computing this structure in case of existence. Obviously, it is assumed that the set S is not stabbed by a line. We distinguish two cases: stabbing wedges W satisfying the separability condition (described in Section 1) or those that do not satisfy such condition. In the first case, we provide an $O(h_S k_S \log n + n \log n)$ time and $O(h_S + n)$ space algorithm. The range for h_S is from $O(1)$ to $O(n^2)$, and the range for k_S is from $O(1)$ to $O(n)$. In the second case, we design an $O(c_S k_S \log n + n \log n)$ time and $O(n)$ space algorithm, with range for c_S in between $O(1)$ and $O(n)$.

We now introduce some useful notation for our purpose. Given a line ℓ and a segment s, we can classify the endpoints of s with respect to ℓ whenever ℓ and the line containing s are not parallel. It suffices to do a parallel sweep with ℓ until it crosses s, leaving one endpoint in ℓ^+, and the other one in ℓ^-. These endpoints are denoted by e^+ and e^-, respectively.

Let W be a stabbing wedge for S. The two rays that form W are denoted by ℓ_1 and ℓ_2, and W is written as $W = \{\ell_1, \ell_2\}$. The line containing ℓ_i for $i = 1, 2$ is denoted by ℓ'_i. The half-planes defined by ℓ_i are written as $\ell_i'^+$ and $\ell_i'^-$. Suppose that ℓ_1 stabs a subset of segments $S_1 \subsetneq S$, $S_1 \neq \emptyset$, and the set $S_2 = S \setminus S_1$ is stabbed by ℓ_2.

3.1.1 Stabbing wedges satisfying the separability condition

Since by definition we consider W as a separator structure, a segment can not be stabbed by both rays.

Let S_1^+ (S_1^-) be the set of endpoints of the segments of S_1 classified as e^+ (e^-) with respect to ℓ'_1. Analogously, S_2^+ (S_2^-) is the set of endpoints of the segments of S_2 classified as e^+ (e^-) with respect to ℓ'_2. Thus, S_1^- and S_2^+ are contained inside the wedge W, and S_1^+ and S_2^- are located outside it. Notice that these assignments $\{+,-\}$ depend on the relative position of the lines ℓ'_1 and ℓ'_2, i.e., on the slope and the aperture angle of the stabbing wedge. We concentrate in a particular case but the remaining constant number of cases can be handle in a similar way.

Lemma 1 *If $W = \{\ell_1, \ell_2\}$ is a stabbing wedge for S, ℓ'_1 and ℓ'_2 are extreme lines for S and at least half of the segments of S are stabbed by either ℓ_1 or ℓ_2.*

The next lemma assumes the following conditions: (i) let ℓ'_1 be an extreme line for S where $S_1 \subsetneq S$, $S_1 \neq \emptyset$, is the subset of segments stabbed by ℓ'_1. Let S_1^+ and S_1^- be the classification of the endpoints of the segments of S_1 given by ℓ'_1, and let $S_2 = S \setminus S_1$. (ii) Let m be a fixed slope and let S_2^+ and S_2^- be the classification of the endpoints of the segments of S_2 by sweeping a line with slope m.

Lemma 2 *There exists a stabbing wedge $W = \{\ell_1, \ell_2\}$ for S with ℓ_1 contained in ℓ'_1 if and only if S_2^- is line separable from $S_1^- \cup S_2^+$. The locus of apices of the stabbing wedges for S respecting this classification of endpoints is a (possible unbounded and degenerate) convex quadrilateral Q defined by the following four lines: the interior supported lines between $CH(S_1^+)$ and $CH(S_1^- \cup S_2^+)$ and the interior supported lines between $CH(S_1^- \cup S_2^+)$ and $CH(S_2^-)$.*

Lemmas 1 and 2 are the key tools to design an algorithm that proves the following result.

Theorem 3 *The set of combinatorially different stabbing wedges for S with the separability condition, together with a representation of them formed by the locus of apices of the stabbing wedges can be computed in $O(h_S k_S \log n + n \log n)$ time and $O(h_S + n)$ space. The problem of deciding whether there exists a stabbing wedge for an arbitrary segment set in the plane has an $\Omega(n \log n)$ time lower bound in the fixed order algebraic decision tree model.*

3.1.2 Stabbing wedges not satisfying the separability condition

Let $W = \{\ell_1, \ell_2\}$ be a stabbing wedge for S not verifying the separability condition. A property that only holds for this type of wedges is the following: *there always exists a stabbing wedge W for S (not satisfying the separability condition) formed by two rays both anchored on fixed points of S, i.e., both rays are contained in critical extreme lines.* This property let us prove the following result.

Theorem 4 *The set of combinatorially different stabbing wedges for S not satisfying the separability condition can be computed in $O(c_S k_S \log n + n \log n)$ time and $O(n)$ space.*

3.2 Stabbing wedges for parallel segments with equal length

Let $S = \{s_1, \ldots, s_n\}$ be a set of n parallel segments in the plane *with equal length* which are not stabbed by a line. We now consider the problem of computing a stabbing wedge W for S satisfying the separability condition.

Up to symmetry with respect to either the x-axis or the y-axis, we distinguish three types of wedges according to the relative position of the rays ℓ_1 and ℓ_2 of the possible stabbing wedge $W = \{\ell_1, \ell_2\}$ for S. Let α_W be the *aperture angle* or *interval direction* defined by the rays ℓ_1 and ℓ_2 of W. The three types are the following: (a) α_W contains the vertical direction; (b) α_W contains the horizontal direction; and (c) both rays ℓ_1 and ℓ_2 of W have positive slope.

3.2.1 Type (a)

When α_W contains the vertical direction, it is possible to design the following $O(n \log n)$ time algorithm for computing a stabbing wedge for S. It is based on an $O(n \log n)$ time and $O(n)$ space algorithm for deciding the wedge separability of a red-blue point set in the plane [6].

1. In $O(n)$ time, classify the endpoints of the segments of S as follows. For each segment s, color red the endpoint of s with bigger y-coordinate and color blue the other endpoint. Let R and B be the sets of red and blue endpoints.

2. In $O(n \log n)$ time, decide whether there exists a separating wedge for R and B, and compute it in case of existence. In the same time, we can compute the locus of apices of all the separating wedges formed by convex quadrilaterals.

Theorem 5 *Given the set S, a stabbing wedge of type (a) for S can be computed in $O(n \log n)$ time and $O(n)$ space.*

If the slopes of the rays of the wedge are known, there exists an $O(n)$ time algorithm to compute a stabbing wedge for S using the median of the x-coordinates of the endpoints of the segments.

3.2.2 Type (b)

We have also obtained an $O(n \log n)$ time algorithm for computing a stabbing wedge for S when α_W contains the horizontal direction. For each segment $s_i \in S$, consider its midpoint ρ_i. In $O(n \log n)$ time, sort these midpoints by decreasing y-coordinate, and let $\preceq_y$ denote this order. Denote by $S^* = \{s_1^*, \ldots, s_n^*\}$ the set of segments of S sorted by the $\preceq_y$ order of their midpoints. The key tools to design our algorithm are given by the following lemma.

Lemma 6 *If there exists a stabbing wedge for S, $W = \{\ell_1, \ell_2\}$, such that α_W contains the horizontal direction, then the midpoints of the segments stabbed by ℓ_1 appear before in the $\preceq_y$ order than the midpoints of the segments stabbed by ℓ_2.*

Let ℓ_1' be a line with positive slope that stabs $S_1 \subsetneq S$, $S_1 \neq \emptyset$, and let ℓ_2' be a line with negative slope that stabs $S_2 = S \setminus S_1$. Consider the classification of the endpoints of S provided by ℓ_1' and ℓ_2'. There exists a stabbing wedge $W = \{\ell_1, \ell_2\}$ for S if and only if both S_1^+ and S_2^- are line separable from $S_1^- \cup S_2^+$.

Theorem 7 *Given the set S, a stabbing wedge of type (b) for S can be computed in $O(n \log n)$ time and $O(n)$ space.*

3.2.3 Type (c)

When both rays ℓ_1 and ℓ_2 of W have positive slope, the main problem is to obtain a consistent classification of the midpoints of the segments according to the possible stabbing wedge of type (c). Denote by d the length of the segments of S, and recall that ρ_i is the midpoint of the segment $s_i \in S$.

Assume that there exists a stabbing wedge of type (c) for S, denoted by $W = \{\ell_1, \ell_2\}$. Suppose also that α_W is known. Let ℓ_i' for $i = 1, 2$, be the line containing the ray ℓ_i. Denote by ℓ_1'' the line below and parallel to ℓ_1', such that the *vertical distance* between the two lines ℓ_1'' and ℓ_1' is exactly $d/2$. Similarly, ℓ_2'' is the line above and parallel to ℓ_2' such that the vertical distance between ℓ_2'' and ℓ_2' is also $d/2$. Let ℓ be the bisector of the angle defined by ℓ_1' and ℓ_2' or any line with slope between the slopes of ℓ_1' and ℓ_2'. Now rotate the coordinate system such that the line ℓ becomes the new x-axis and proceed as in Theorem 7. Thus, if we know that $\alpha_W \geq \alpha$, for some given α, we can compute a constant number $2\pi/\alpha = t$ of slope candidates for line ℓ and check each one in $O(n \log n)$ time and $O(n)$ space. For stabbing wedges with very small aperture angle α_W, the value t can dominate n.

Theorem 8 *Given the set S and a positive value α, a stabbing wedge of type (c) for S with $\alpha_W \geq \alpha$ can be computed in $O(nt \log n)$ time and $O(n)$ space.*

4 Other stabbers

Table 1 summarizes the times and the spaces complexities of the algorithms that we have obtained for the stabbers we consider in this paper. The notations

Stabber	**Time**	**Space**
Wedge (SC)	$O(h_S k_S \log n + n \log n)$	$O(h_S + n)$
Wedge (NSC)	$O(c_S k_S \log n + n \log n)$	$O(n)$
Zig-zag (SC)	$O(n^2 k_S \log n + n \log n)$	$O(h_S + n)$
Zig-zag (NSC)	$O(c_S^2 k_S \log n + n \log n)$	$O(n)$
Double-wedge	$O(n^2 k_S \log n)$	$O(n^2)$
2-level tree	$O(n^2 {k_S}^2 \log n)$	$O(n^2)$

Table 1: Summary of results

(SC) and (NSC) mean *satisfying and not satisfying the separability condition.* These two possibilities are only considered for the stabbing wedge and the stabbing zig-zag. To conclude, we want to stress that in case of existence, our algorithms compute all the combinatorially different solutions.

References

[1] D. Avis, J.-M. Robert, and R. Wenger. Lower bounds for line stabbing. *Inform. Process. Lett.*, 33, (1989), 59–62.

[2] M. Claverol. Problemas geométricos en morfología computacional. *PhD Thesis. Universitat Politècnica de Catalunya*, 2004.

[3] H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, Vol. 22, (1982) pp. 274–281.

[4] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The upper envelope of piecewise linear functions: algorithms and applications. *Report No. UIUCDCS-R-87-1390*, University of Ilinois, (1987).

[5] M. T. Goodrich and J. S. Snoeyink. Stabbing parallel segments whit a convex polygon. *Workshop Algorithms Data Struct. Lecture Notes Comput. Sci.*, 382 (1989), 231–242.

[6] F. Hurtado, M. Noy, P. A. Ramos, and C. Seara. Separating objects in the plane by wedges and strips. *Discrete Applied Mathematics*, Vol. 109, (2000), pp. 109–138.

[7] K. A. Lyons, Henk Meijer, and David Rappaport. Minimum polygon stabbers of isothetic line segments. *Department of Computing and Information Science*, Queen's University, Canada, (1990).

[8] J. O'Rourke. An online algorithm for fitting straigt lines between data ranges. *CACM*, 24, (1981), 574–578.

[9] D. Rappaport. Minimum polygon transversals of line segments. *International Journal of Computational Geometry & Applications*, Vol. 5, No. 3, (1995), 243–256.

Online Square Packing

Sándor P. Fekete* Tom Kamphans* Nils Schweer*

Abstract

We analyze the problem of packing squares in an online fashion: Given an semi-infinite strip of width 1 and an unknown sequence of squares with side lengths in $[0, 1]$ that arrive from above, one at a time. The objective is to pack these items as they arrive, minimizing the resulting height. Just like in the classical game of Tetris, each square must be moved along a collision-free path to its final destination; in addition, we may have to account for gravity in both motion and position (i.e, squares are not allowed to move up and any final destination has to be supported from below). This problem has been considered before; the best previous result is by Azar and Epstein, who gave a 4-competitive algorithm in a setting without gravity, based on ideas of shelf-packing, with the possibility of letting squares "hang in the air" in order to assign them to different levels, allowing an analysis that is reminiscent of some bin-packing arguments.

We present an algorithm with competitive factor $\frac{34}{13} \approx 2.6154$, with or without the presence of gravity.

1 Introduction

Packing problems arise in many different situations, either concrete (where actual physical objects have to be packed), or abstract (where the space is virtual, e.g., in scheduling). Even in a one-dimensional setting, computing an optimal set of positions in a container for a known set of objects is a classical, hard problem. Having to deal with two-dimensional objects adds a variety of difficulties; one of them is the more complex structure of feasible placements; see, for example, Fekete et al. [8]. Another one is actually moving the objects into their final locations without causing collisions or overlap along the way.

A different kind of difficulty may arise from a lack of information: in many settings, objects have to be assigned to their final locations one by one, without knowing future items. Obviously, this makes the challenge even harder.

In this paper, we consider online packing of squares into a vertical strip of unit width. Squares arrive from above in an online fashion, one at a time, and have to be moved to their final positions. On this path, a square may move only through unoccupied space; in allusion to the well-known computer game, this is called the *Tetris constraint*. In addition, an item is not allowed to move upwards and has to be supported from below when reaching its final position; these conditions are called *gravity constraints*. The objective is to minimize the total height of the occupied part of the strip.

*Braunschweig University of Technology, Computer Science, Algorithms Group, 38106 Braunschweig, Germany

1.1 Related Work

There is a considerable amount of work of online rectangle packing without the Tetris constraint, see Coffman et al. [4] for a quick overview. Most notably, shelf-packing approaches [2] (in which the strip is subdivided into shelves of certain heights, and no two rectangles within the same shelf get placed on top of each other) cannot do better than an asymptotic worst-case ratio of 1.691..., which can be achieved by Harmonic shelf packing, see Csirik and Woeginger [5].

Every reader is certainly familiar with the classical game of Tetris: Given a strip of fixed width, find online placements for a sequence of objects falling down from above, such that space is utilized as best as possible[1]. In this process, no item can ever move upward or collide with another object. An item will come to a stop only if it is supported from below, and each placement has to be fixed before the next item arrives. Even when disregarding the difficulty of ever-increasing speed, Tetris is notoriously difficult: As was shown by Breukelaar et al. [3], Tetris is PSPACE-hard, even for the original, limited set of different objects.

Tetris-like online packing has been studied before. Most notably, Azar and Epstein [1] considered online packing of rectangles into a strip; just like in Tetris, they considered the situation with or without rotation of objects. For the case without rotation, they showed that no constant competitive ratio is possible, unless there is a fixed-size lower bound of ε on the size of the objects, in which case there is an upper bound of $O(\log \frac{1}{\varepsilon})$. For the case in which rotation is possible, they showed a 4-competitive strategy, based on shelf-packing methods, with all rectangles being rotated to be placed on their narrow sides; until now, this is also the best deterministic upper bound for squares.

[1]Obviously, there is a slight difference in the objective function, because Tetris aims at filling rows. In actual optimization scenarios, this is less interesting, as it is not critical whether a row is used to precisely 100%—in particular, as full rows do not magically disappear in real life.

In this strategy, gravity is not taken into account, as items are allowed to be placed at appropriate levels, even if they are unsupported.

More recently, Coffmann, Downey, and Winkler [4] considered probabilistic aspects of online rectangle packing with Tetris constraint, without allowing rotations. If n rectangle sizes are chosen uniformly at random from the interval $[0, 1]$, they showed that there is a lower bound of $(0.31382733...)n$ on the expected height of the strip; using another kind of level-type strategy, which arises from the bin-packing–inspired *Next Fit Level*, they established an upper bound of $(0.36976421...)n$ on the expected height. Epstein and van Stee gave an optimal bounded space algorithm for online hypercube bin-packing for any dimension $d \geq 2$, and unbounded-space, competitive algorithms for square and cube packing [6, 7].

2 Problem and Definitions

We are given a strip, S, of width 1 which is closed at the bottom and has infinite height, as well as a sequence, $A_1, \dots, A_n$, of squares with side length $|A_i| \in [0, 1]$. The squares are presented one by one; the next square always arrives above all previously placed squares. Our goal is to find a nonoverlapping placement of squares in the strip that keeps the height of the occupied area as low as possible. The sides of the squares in the placement are parallel to the sides of the strip. A packing has to fulfill two additional constraints:

Gravity constraint: A square must be packed on top of a square that has been packed before (i.e., the intersection of the upper square's bottom and the lower square's top must be a line segment); in addition, no square may ever move up.

Tetris constraint: At the time a square is placed, there is a collision-free path from the top of the strip to the square's final position.

We consider the online problem; that is, the sequence $\mathcal{A}$ is not known in advance. Our strategy gets the squares one by one and has to place a square before it gets the next.

We call a square A_j a *bottom neighbor* of a square A_i if the top side of A_j and the bottom side of A_i overlap in more than one point. For every square A_i we define the *bottom sequence* as follows: A_i is the first element of this sequence and the next element is chosen as an arbitrary bottom neighbor of the previous element. The sequence ends if no such neighbor exists.

3 Algorithm

Consider two vertical lines of infinite length going upwards from the bottom side of S and parallel to the left and the right side of S. We call the area between these lines a *slot*, the lines the *left boundary* and the *right boundary* of the slot, and the distance between the lines the *width* of the slot.

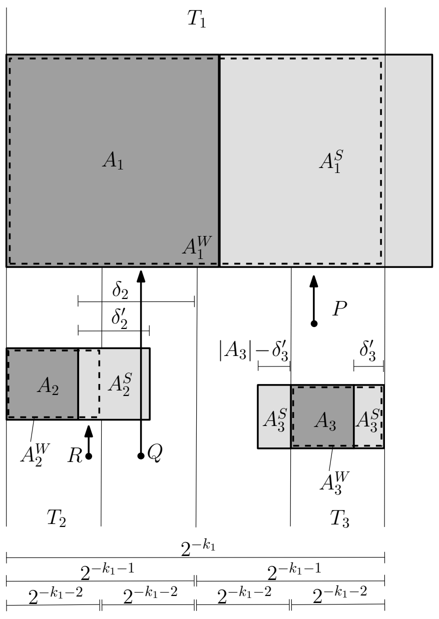


Figure 1: Squares A_i with their *shadows* A_i^S and their *widening* A_i^W. δ_2' is equal to $|A_2|$ and δ_3' is equal to δ_3. The points P and Q are charged to A_1. R is not charged to A_1, but to A_2.

Now our algorithm works as follows: We divide the strip S of width 1 into slots of different widths; for every $j = 0, 1, 2 \dots$ we create 2^j slots of width $\frac{1}{2^j}$ side by side; that is, we divide S into one slot of width 1, two slots of width $\frac{1}{2}$, four slots of width $\frac{1}{4}$ and so on. Note that a slot of width 2^{-i} contains 2 slots of width 2^{-i-1}; see Fig. 1.

For every square A_i we round the side length $|A_i|$ to the smallest number $\frac{1}{2^{k_i}}$ that is larger than or equal to $|A_i|$. We place A_i in the slot of width 2^{-k_i} that allows A_i to be placed as near to the bottom of S as possible by moving A_i down along the left boundary of the chosen slot until another square is reached. We call this algorithm *SlotAlgorithm*. It clearly satisfies the Tetris and the Gravity constraints and next we show that the produced height is at most 2.6154 times the height of an optimal packing.

4 Analysis

Let A_i be a square placed by the SlotAlgorithm in a slot T_i of width 2^{-k_i}. Let δ_i be the distance between the right side of A_i and the right boundary of the slot of width 2^{-k_i+1} that contains A_i and $\delta_i' := \min\{|A_i|, \delta_i\}$. We call the area obtained by enlarging A_i by δ_i' to the right and by $|A_i| - \delta_i'$ to the left the *shadow* of A_i and denote it by A_i^S. Thus, A_i^S is an area of the same size as A_i and lies completely inside a slot of twice the width of A_i's slot. Moreover, we define the *widening* of A_i as $A_i^W = (A_i \cup A_i^S) \cap T_i$; see Fig. 1.

Now, consider a point P in T_i that is not inside an A_j^W for any square A_j and that does not lie on the left or the right boundary of any slot. We charge P to the square A_i if A_i^W is the first widening that intersects the vertical line going upwards from P. We denote by F_{A_i} the set of all points charged to A_i and by $|F_{A_i}|$ its area. The set of points lying on the left or the right boundary of any slot has area zero and is therefore neglected in the rest of this section. For the analysis, we place a closing square, A_{n+1}, of side length 1 on top of the packing.[2] Therefore, every point in the packing that does not lie inside an A_j^W is charged to a square. Because A_i and A_i^S have the same area, we can bound the height, ALG, of the packing produced by the algorithm as follows:

$$ALG \leq 2\sum_{i=1}^{n} |A_i|^2 + \sum_{i=1}^{n+1} |F_{A_i}|$$

Theorem 1 *The SlotAlgorithm is competitive with factor 2.6154.*

Proof. The height of an optimal packing is at least $\sum_{i=1}^{n} |A_i|^2$ and, therefore, it suffices to show that $|F_{A_i}| \leq 0.6154 \cdot |A_i|^2$ holds for every square A_i. We construct for every A_i a sequence of squares $B_1^i, B_2^i, \ldots, B_m^i$ with $B_1^i = A_i$ (to ease notation, we omit the superscript i in the following). We denote by E_{B_j} the extension of the bottom side of B_j to the left and to the right (Fig. 2). We will show that by an appropriate choice of the sequence we can bound the area of the part of F_{B_1} that lies between a consecutive pair of extensions, E_{B_j} and $E_{B_{j+1}}$, in terms of B_{j+1} and the slot widths. From this we will derive the upper bound on the area of F_{B_1}. We assume throughout the proof that the square B_j, $j \geq 1$, is placed in a slot, T_j, of width 2^{-k_j}. Note that F_{B_1} is completely contained in T_1.

A slot is called *active (with respect to E_{B_j} and B_1)* if there is a point in the slot that lies below E_{B_j} and that is charged to B_1 and *nonactive* otherwise. If it is clear from the context we leave out the B_1.

The sequence of squares is chosen as follows: B_1 is the first square and the square B_{j+1}, $j = 1, \ldots, m-1$ is chosen as the smallest one that intersects or touches E_{B_j} in an active slot (w.r.t. E_{B_j} and B_1) of width 2^{-k_j} and that is not equal to B_j. The sequence ends if all slots are nonactive w.r.t. to an extension E_{B_m}. We claim the following:

(i) B_{j+1} exists for $j+1 \leq m$ and $|B_{j+1}| \leq 2^{-k_j-1}$ for $j+1 \leq m-1$.

[2]This does not affect the competitive factor asymptotically.

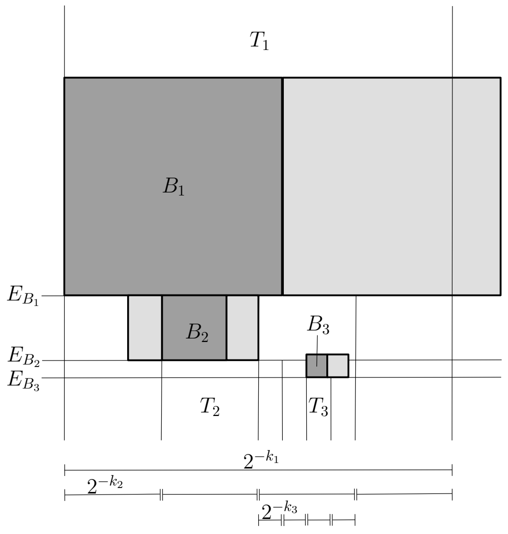


Figure 2: The first three squares of the sequence. In this example, B_2 is the smallest square that bounds B_1 from below. B_3 is the smallest one that intersects E_{B_2} in an active slot (w.r.t. E_{B_2}) of width $\frac{1}{2^{k_2}}$. T_2 is nonactive (w.r.t. E_{B_2}) and, of course, also w.r.t. all extension E_{B_j}, $j \geq 3$

(ii) The number of active slots (w.r.t. E_{B_j}) of width 2^{-k_j} is at most

$$\begin{cases} 1 & \text{, for } j = 1 \\ \prod_{i=2}^{j} (\frac{1}{2^{k_{i-1}}} 2^{k_i} - 1) & \text{, for } j \geq 2 \end{cases}$$

(iii) The area of the part of F_{B_1} that lies in an active slot of width 2^{-k_j} between E_{B_j} and $E_{B_{j+1}}$ is at most $2^{-k_j}|B_{j+1}| - 2|B_{j+1}|^2$.

We prove the claims by induction. If B_1 is placed on the bottom of S, F_{B_1} has size 0 and B_1 is the last element of the sequence. Otherwise, the square B_1 has at least one bottom neighbor, which is a candidate for the choice of B_2. If $|B_2| > 2^{-k_1-1}$, then B_2 is a bottom neighbor of B_1 of side length greater than or equal to B_1 and, thus, all points below are charged to B_2. Hence, slot T_1 is nonactive and F_{B_1} is of size zero.

If (i) is fulfilled T_1 is the only slot of width 2^{-k_1} that is active. Moreover, we conclude that the area of the part of F_{B_1} that lies between E_{B_1} and E_{B_2} is at most $2^{-k_1}|B_2| - 2|B_2|^2$ (Fig. 2). Note that we can subtract the area of B_2 twice, because B_2^S was defined to lie completely inside a strip of width $2^{-k_2+1} \leq 2^{-k_1}$ and is of same area as B_2.

Now suppose for a contradiction that the $(j+1)$th element does not exist for $j+1 \leq m$. Let T' be an active slot in T_1 (w.r.t. E_{B_j}) of width 2^{-k_j} where E_{B_j} is not intersected by a square in T'. If there is an ε such that for every point $P \in (T' \cap E_{B_j})$ there is a point P' at a distance of ε below P that is charged to B_1 we

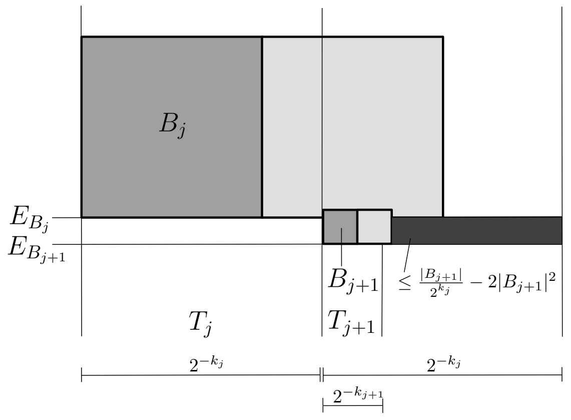


Figure 3: The part of F_{B_1} (darkest gray) that lies between E_{B_j} and $E_{B_{j+1}}$ in an active slot of width $\frac{1}{2^{k_j}}$ is at most $\frac{1}{2^{k_j}}|B_{j+1}| - 2|B_{j+1}|^2$ because points in B^W_{j+1} are not charged to B_1.

conclude that there would have been a better position for B_j. Hence, there is at least one point, Q, below E_{B_j} that is not charged to B_1. Consider the bottom sequence of the square Q is charged to. This sequence has to intersect E_{B_j} outside of T' (by choice of T'). But then one of its elements has to intersect the left or the right boundary of T' and we can conclude that this square has at least the width of T', because (by the algorithm) a square with rounded side length $2^{-\ell}$ cannot cross a slot's boundary of width larger than $2^{-\ell}$. In turn, a square larger than T' completely covers T' and T' cannot be active w.r.t. to E_{B_j} and B_1. Thus, all points in T' below E_{B_j} are charged to this square; a contradiction. This proves the existence of B_{j+1}. Because we chose B_{j+1} to be of minimal side length, $|B_{j+1}| \geq 2^{-k_j}$ would imply that all slots inside T are nonactive (w.r.t. E_{B_j}). Therefore, if B_{j+1} is not the last element of the sequence, $|B_{j+1}| \leq 2^{-k_j-1}$ holds.

By the induction hypothesis there are at most $(\frac{1}{2^{k_1}}2^{k_2} - 1) \cdot (\frac{1}{2^{k_2}}2^{k_3} - 1) \cdot \ldots \cdot (\frac{1}{2^{k_{j-2}}}2^{k_{j-1}} - 1)$ active slots of width $2^{-k_{j-1}}$ (w.r.t. $E_{B_{j-1}}$). Each of these slots contains $2^{k_j - k_{j-1}}$ slots of width 2^{-k_j} and in every active slot of width $2^{-k_{j-1}}$ at least one slot of width 2^{-k_j} is nonactive because we chose B_j to be of minimum side length. Hence, the number of active slots (w.r.t. E_{B_j}) is a factor of $(\frac{1}{2^{k_{j-1}}}2^{k_j} - 1)$ larger than the number of active slots (w.r.t. $E_{B_{j-1}}$).

Again by the choice of B_{j+1} and by the fact that in every active slot of width 2^{-k_j} there is at least one square, B, intersecting E_{B_j} (points below B^W are not charged to B_1) we conclude that the area of F_{B_1} between E_{B_j} and $E_{B_{j+1}}$ is at most $2^{-k_j}|B_{j+1}| - 2|B_{j+1}|^2$ in every active slot of width 2^{-k_j} (Fig. 3).

Altogether, we get an upper bound on $|F_{B_1}|$ of

$$\frac{|B_2|}{2^{k_1}} - 2|B_2|^2 + \sum_{j=2}^{m}\left[\left(\frac{|B_{j+1}|}{2^{k_j}} - 2|B_{j+1}|^2\right)\prod_{i=1}^{j-1}\left(\frac{2^{k_{i+1}}}{2^{k_i}} - 1\right)\right]$$

This expression is maximized if we choose $|B_{i+1}| = \frac{1}{2^{k_i+2}}$ for $i = 1, \ldots, m$. This implies $k_i = k_1 + 2(i-1)$ and we get the upper bound

$$|F_{B_1}| \leq \sum_{i=0}^{\infty} \frac{3^i}{2^{2k_1+4i+3}} .$$

The fraction $\frac{|F_{B_1}|}{|B_1|^2}$ is maximized if we choose $|B_1|$ as small as possible; that is, $B_1 = 2^{-(k_1+1)} + \varepsilon$. We conclude:

$$\begin{aligned}\frac{|F_{B_1}|}{|B_1|^2} &\leq \sum_{i=0}^{\infty}\frac{2^{2k_1+2}\cdot 3^i}{2^{2k_1+4i+3}} = \sum_{i=0}^{\infty}\frac{3^i}{2^{4i+1}} \\ &= \frac{1}{2}\cdot\sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i = \frac{8}{13} = 0.6154...\end{aligned}$$

□

5 Conclusion

We presented an algorithm that is 2.6154-competitive. We believe that our algorithm can be improved; at this point, the best known lower bound is 1.25. We believe that our approach can be extended to higher dimensions. Rectangles may require a slightly different analysis. These topics will be the subject of future research. It is an open question, whether our analysis is tight or can be improved. The best lower bound for *SlotAlgorithm* known to us is 2.

References

[1] Y. Azar and L. Epstein. On two dimensional packing. *J. Algorithms*, 25:290–310, 1997.

[2] B. S. Baker and J. S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM J. Comput.*, 12:508–525, 1983.

[3] R. Breukelaar, E. D. Demaine, S. Hohenberger, H. J. Hoogeboom, W. A. Kosters, and D. Liben-Nowell. Tetris is hard, even to approximate. *Internat. J. Comput. Geom. Appl.*, 14:41–68, 2004.

[4] E. G. Coffman Jr., P. J. Downey, and P. Winkler. Packing rectangles in a strip. *Acta Inform.*, 38:673–693, 2002.

[5] J. Csirik and G. J. Woeginger. Shelf algorithms for online strip packing. *Inform. Proc. Let.*, 63:171–175, 1997.

[6] L. Epstein and R. van Stee. Optimal online bounded space multidimensional packing. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 214–223, 2004.

[7] L. Epstein and R. van Stee. Online square and cube packing. *Acta Inform.*, 41:595–606, 2005.

[8] S. P. Fekete, J. Schepers, and J. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55:569–587, 2007.

Square and Rectangle Covering with Outliers

Hee-Kap Ahn[§] Sang Won Bae[†] Sang-Sub Kim[§] Matias Korman[‡] Iris Reinbacher[†]
Wanbin Son[§]

Abstract

For a set of n points in the plane, we consider the axis–aligned (p,k)-Box Covering problem: Find p axis–aligned, pairwise disjoint boxes that together contain exactly $n-k$ points, (i.e., there are k outliers that are not covered by any box). In this paper, we allow the boxes to be either squares or rectangles, and we want to minimize the area of the largest box. For squares, we present algorithms that find the solution in time $O(n + k\log k)$ for $p = 1$, and in time $O(n\log n + k^p \log^p k)$ for $p = 2,3$. For rectangles, the running times increase by a factor of $k^2/\log k$ in the second term for all cases. In all cases, our algorithms use $O(n)$ space.

1 Introduction

Given a set P of n points in the plane, we consider the (p,k)-Box Covering problem: Find p pairwise disjoint boxes that together contain $n-k$ points of P, such that the area of the largest box is minimized. We will refer to the k points that are not contained in the union of the boxes as *outliers*, and we will consider the boxes to be axis parallel squares or rectangles. We assume that there are no two points with the same x- or y-coordinates. Previous work on this problem has mainly focused on the problem without outliers. The $(p,0)$-Box Covering problem is closely related to the (weighted) rectilinear p-center problem where p squares may have different sizes and are allowed to overlap. This problem is of LP–type [10], and thus can be solved in linear time for $p \le 3$ [5]. Several papers consider variations of the problem without outliers, and achieve different running times, see e.g., [6, 8, 9].

There is less work when outliers are allowed, most of it considers the $(1,k)$-Box Covering problem. Agarwal et al. [1] achieved a running time of $O((n-k)^2 n\log n)$ using $O((n-k)n)$ space for the $(1,k)$-Box Covering, where the box is an axis–parallel square or rectangle. A randomized algorithm that runs in $O(n\log n)$ time was later given for the $(1,k)$-Square Covering problem by Chan in [3].

We improve the previous results by giving algorithms that run in $O(n + k\log k)$ time for the $(1,k)$-Square Covering problem and in $O(n + k^3)$ time for the $(1,k)$-Rectangle Covering. We use these algorithms as the base for recursion to compute the (p,k)-Box Covering for $p \le 3$. The running time of our algorithms can be seen in Table 1, they all use $O(n)$ space.

[†]Division of Computer Science, KAIST, South Korea. {swbae, iris}@tclab.kaist.ac.kr
[‡]Graduate School of Information Sciences, Tohoku University, Japan. mati@dais.is.tohoku.ac.jp
[§]Department of Computer Science and Engineering, POSTECH, South Korea. {heekap, helmet1981, mnbiny}@postech.ac.kr

2 Square Covering

2.1 $(1,k)$-Square Covering

Before focusing on our algorithm, we mention the previously fastest algorithm for the $(1,k)$-Square Covering problem:

Theorem 1 ([3]) *Given a set P of n points in the plane, the $(1,k)$-Square Covering problem can be solved in $O(n\log n)$ expected time using $O(n)$ space.*

There is a deterministic version of the same algorithm [7] that runs $O(n\log^2 n)$ time. As we use the above algorithm as base case for recursion, our algorithm is randomized (if k is large). However, our algorithms can be transformed into deterministic ones at the cost of an additional $O(\log k)$ factor in the second term.

A point $p \in P$ is called $(k+1)$-*extreme* if either its x- or y-coordinate is among the $k+1$ smallest or largest in P. Let $E(P)$ be the set of $(k+1)$-extreme points of P.

Lemma 2 *For a given set P of n points in the plane, we can compute the set $E(P)$ of all $(k+1)$-extreme points of P in $O(n)$ time.*

Proof. We first select the point p_L (p_R, resp.) of P with $(k+1)$-th smallest (largest, resp.) x-coordinate in linear time using a selection algorithm. We go through P again to find all points with x-coordinates smaller (larger, resp.) than p_L (p_R, resp.). Similarly, we select points according to their y-coordinates. $\square$

It is easy to see that the left side of the optimal solution of the $(1,k)$-Square Covering problem lies

Table 1: Running times of our (p, k)-Box Covering algorithms

No. of boxes	Squares	Rectangles
$p = 1$	$O(n + k \log k)$	$O(n + k^3)$
$p = 2$	$O(n \log n + k^2 \log^2 k)$	$O(n \log n + k^4 \log k)$
$p = 3$	$O(n \log n + k^3 \log^3 k)$	$O(n \log n + k^5 \log k)$

on or to the left of the vertical line through p_L, and that the right side lies on or to the right of the vertical line through p_R (otherwise there are more than k outliers). A similar observation holds for the top and bottom sides of the optimal solution, and we get:

Lemma 3 *The optimal square B^* that solves the $(1, k)$-*Square Covering *problem is determined by the points of $E(P)$ only.*

With this observation we can improve the running time of the algorithm from Theorem 1 as follows:

Theorem 4 *Given a set P of n points in the plane, the $(1, k)$-*Square Covering *problem can be solved in $O(n + k \log k)$ time using $O(n)$ space.*

Proof. We first compute the set of extreme points $E(P)$ using Lemma 2 and then use the algorithm from Theorem 1 on the set $E(P)$. The time bound follows from $|E(P)| \leq 4k + 4$. □

We present a lower bound of the problem:

Lemma 5 *If k is part of the input, for any n points there is an $\Omega(n \log n)$ lower bound on computing the minimal enclosing p squares or rectangles in the algebraic decision tree model.*

Proof. We reduce from the set disjointness problem, which is to decide whether or not there is a repeated element in a given sequence of n points in $\mathbb{R}^2$. An $\Omega(n \log n)$ bound of set disjointness in the algebraic decision tree model was given in [11] (see Cor. 5.1). Thus, given any sequence $p_1, \ldots, p_n$, we compute the minimal enclosing p squares allowing $k = n - p - 1$ outliers, i.e., the union of the p squares must cover exactly $p + 1$ points.

Hence, the enclosing squares will have area zero (i.e., they are degenerate squares of side length zero) if there is a repeated element in the sequence. Otherwise, by the pigeonhole principle, one of the squares must cover two points and will have a positive area. A similar proof holds for rectangles. □

2.2 $(2, k)$-Square Covering

The following observation is crucial to solve the $(2, k)$-Square Covering problem:

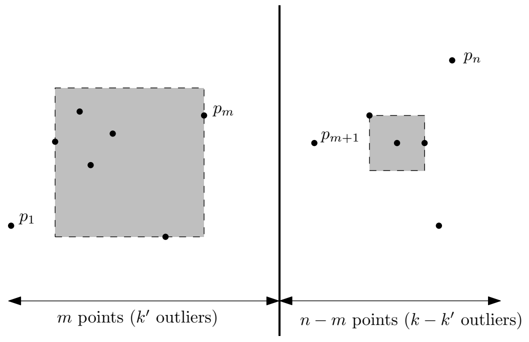

p_n
p_m
p_{m+1}
p_1
m points (k' outliers)
$n - m$ points ($k - k'$ outliers)

Figure 1: For given k' and m, the optimal m^* lies on the side with the larger square (left in this example).

Observation 1 *For any two pairwise disjoint axis-aligned squares in the plane, there exists an axis-parallel line ℓ that separates them.*

Hence, there is always an axis-parallel line ℓ that separates the two optimal squares (B_1^*, B_2^*) of the $(2, k)$-Square Covering problem. Let ℓ^+ be the halfplane defined by ℓ that contains B_1^*. Let P^+ be the set of points of P that lie in ℓ^+ (including points on ℓ), and let k^+ be the number of outliers of the solution of the $(2, k)$-Square Covering problem that lie in ℓ^+. Then there is always an optimal solution of the $(1, k^+)$-Square Covering problem for P^+ with size smaller than or equal to that of B_1^*. The same argument also holds for the other halfplane ℓ^-, B_2^*, and k^-, where $k^- = k - k^+$. Thus, the pair of solutions of the two $(1, k')$-Square Covering problems, for $k' = k^+$ and $k' = k^-$, is also an optimal solution of the original $(2, k)$-Square Covering problem.

Lemma 6 *There exists an axis-parallel line ℓ and a positive integer $k' \leq k$ such that the pair of optimal solutions of the $(1, k')$-*Square Covering *problem restricted to ℓ^+ and of the $(1, k - k')$-*Square Covering *problem restricted to ℓ^- is an optimal solution of the $(2, k)$-*Square Covering *problem of $\ell^+ \cup \ell^-$.*

We assume w.l.o.g. that ℓ is vertical, and we associate ℓ with m, the number of points that lie to the left of (or on) ℓ. Let $p_1, p_2, \ldots, p_n$ be the list of points in P sorted by x-coordinate. Then ℓ partitions the points of P into two subsets, a left point set, $P_L(m) = \{p_1, \ldots, p_m\}$ and a right point set $P_R(m) = \{p_{m+1}, \ldots, p_n\}$, see Figure 1. Then the optimal left square is a solution of the $(1, k')$-Square Covering problem for $P_L(m)$ for $0 \leq k' \leq k$, and the optimal right square is a solution of the $(1, k-k')$-Square Covering problem for $P_R(m)$.

We can efficiently compute the optimal solutions for $P_L(m)$ and $P_R(m)$ on each halfplane of a vertical line ℓ using the above $(1, k)$-Square Covering algorithm. However, as we will consider many partitioning lines, it is important to find an efficient way to compute the $(k+1)$-extreme points for each $P_L(m), P_R(m)$ corresponding to a particular line ℓ. In order to do so, we will use Chazelle's *Segment Dragging Query* algorithm [4] to compute the $(k+1)$-extreme points.

Lemma 7 ([4]) *Given a set P of n points in the plane, we can preprocess it in $O(n \log n)$ time and $O(n)$ space such that, for any axis–aligned orthogonal range query Q, we can find the point $p \in P \cap Q$ with highest y-coordinate in $O(\log n)$ time.*

Using the above algorithm $k+1$ times allows us to find the $k+1$ points with highest y coordinate in any halfplane. Repeating the same process for rotated versions of P leads to the following corollary.

Corollary 8 *After $O(n \log n)$ preprocessing time, we can compute $E(P_L(m))$ and $E(P_R(m))$ in $O(k \log n)$ time for any given m.*

Before presenting our algorithm we need the following lemma:

Lemma 9 *For a fixed k', the size of the solution of the $(1, k')$-Square Covering problem for $P_L(m)$ is an increasing function of m.*

Proof. Consider the set $P_L(m+1)$ and the optimal square B of the $(1, k')$-Square Covering problem for $P_L(m+1)$. Clearly, $P_L(m+1)$ is a superset of $P_L(m)$, as it contains one more point p_{m+1}. Since k' is fixed, the square B has k' outliers in $P_L(m+1)$. If the interior of B intersects the vertical line ℓ through p_m, we translate B horizontally to the left until it stops intersecting ℓ. Let B' be the translated copy of B, then B' lies in the left halfplane of ℓ and there are at most k' outliers to B' among the points in $P_L(m)$. Therefore we can shrink or translate B' and get a square lying in the left halfplane of ℓ that has exactly k' outliers and a size smaller or equal to that of B. This implies that the optimal square for $P_L(m)$ has a size smaller or equal to that of B. ☐

Lemma 9 immediately implies the following:

Corollary 10 *Let (B_1, B_2) be the solution of the $(2, k)$-Square Covering problem with a separating line ℓ with index m and let k' be the number of outliers permitted on the left halfplane of ℓ. Then there is always an optimal separating line ℓ^* with index m^* such that $m^* \leq m$ if the left square B_1 is larger than the right square B_2, and $m^* \geq m$ otherwise.*

The algorithm to solve the $(2, k)$-Square Covering problem is as follows. We start with the vertical line ℓ at the median of the x-coordinates of all points in P. Let k' be the number of outliers permitted on the left halfplane of ℓ. For a given m, we first compute the sets $E(P_L(m))$ and $E(P_R(m))$. Then we use these sets in the recursive call to the $(1, k')$-Square Covering problem for $P_L(m)$ and the $(1, k-k')$-Square Covering problem for $P_R(m)$, respectively, and solve the subproblems independently. The solutions of these subsets give the first candidate for the solution of the $(2, k)$-Square Covering problem, and we now compare the areas of the two obtained squares. Corollary 10 tells us that the optimal index $m^* \leq m$ if the left square has larger area, and $m^* \geq m$ otherwise (see Figure 1). This way, we can use binary search to find the optimal index m^* for the given k'. As the optimal value of k' is unknown, we need to do this for every k'. Finally, we also need to examine horizontal separating lines ℓ by reversing the roles of x- and y-coordinates.

Theorem 11 *For a set P of n points in the plane, we can solve the $(2, k)$-Square Covering problem in $O(n \log n + k^2 \log^2 k)$ time using $O(n)$ space.*

Proof. The above algorithm needs $O(n \log n)$ preprocessing time and an additional $O(k^2 \log^2 n)$ time to obtain the optimal k' and m^* ($O(k \log n)$ different queries, each one needing $O(k \log n)$ time), giving a total running time of $O(n \log n + k^2 \log^2 n)$. By distinguishing between $k^4 \leq n$ and $k^4 > n$, we can reduce the running time to $O(n \log n + k^2 \log^2 k)$. We omit details due to lack of space. ☐

2.3 (p, k)-Square Covering for $p \geq 3$

The above solution for the $(2, k)$-Square Covering problem suggests a recursive approach for the general (p, k)-Square Covering case: Find an axis-parallel line that separates one square from the others and recursively solve the induced subproblems. We can do this for $p = 3$, as Observation 1 can be generalized.

W.l.o.g. we assume that the separating line ℓ is vertical and that the left halfplane only contains one square. Since Corollary 10 can be generalized to $(3, k)$-Square Covering, we solve this case as before: fix the amount of outliers permitted on the left halfplane to k' and iterate k' from 1 to k to obtain the optimal k^*. For each possible k', we recursively solve the two subproblems and use it to obtain the optimal index m such that the area of the largest square is minimized. Preprocessing for this case is sorting the points of S in both x- and y- coordinates and computing the segment dragging query structure.

Each $(2, k-k')$-Square Covering algorithm is executed as described above, except that preprocessing in the recursive steps is no longer needed: The segment dragging queries can be performed since the preprocessing has been done in the higher level. Also, for the binary search, we can use the sorted list of all points in P, which is a superset of $P_R(m)$.

This algorithm leads to a total time complexity of $O(n \log n + k^3 \log^3 n)$, which can be again reduced by distinguishing between $k^6 \leq n$ and $k^6 > n$.

Theorem 12 *For a set P of n points in the plane, we can solve the $(3,k)$-Square Covering problem in $O(n \log n + k^3 \log^3 k)$ time using $O(n)$ space.*

Unfortunately our algorithm does not extend to $p \geq 4$, as Observation 1 does no longer hold in the general case.

3 Rectangle Covering

In this section, we look at the (p,k)-Rectangle Covering problem. As before, we will focus on the $p \leq 3$ case. It is easy to see that Lemma 3 directly extends to rectangles. However, the best we can do is testing all possible rectangles that cover exactly $n-k$ points. Our approach is brute force: We store the points of $E(P)$ separately in four sorted lists, the top $k+1$ points in $T(P)$, the bottom $k+1$ points in $B(P)$, and the left and right $k+1$ points in $L(P)$, and $R(P)$, resp. Note that some points may belong to more than one set.

We first create a vertical slab by drawing two vertical lines through one point of $L(P)$ and $R(P)$ each. All k' points outside this slab are obviously outliers, which leads to $k-k'$ outliers that are still permitted inside the slab. We now choose two horizontal lines through points in $T(P)$ and $B(P)$ that lie inside the slab, such that the rectangle that is formed by all four lines admits exactly k outliers. Inside each of the $O(k^2)$ vertical slabs, there are at most k horizontal line pairs we need to examine, hence we can find the smallest rectangle covering $n-k$ points in $O(k^3)$ time when the sorted lists of $E(P)$ are given. This preprocessing takes $O(n + k \log k)$ time.

Note that this approach leads to the same running time we would get by simply bootstrapping any other existing rectangle covering algorithms [1, 12] to the set $E(P)$, which has also been done recently in [2].

The (p,k)-Rectangle Covering problem for $p = 2, 3$ can be solved with the same recursive approach as for the squares, as Observation 1 readily generalizes here. We conclude with the following theorem:

Theorem 13 *Given a set P of n points in the plane, the $(1,k)$-Rectangle Covering problem can be solved in $O(n + k^3)$ time using $O(n)$ space, the $(2,k)$-Rectangle Covering problem in $O(n \log n + k^4 \log k)$ time and the $(3,k)$-Rectangle Covering problem in $O(n \log n + k^5 \log^2 k)$ time. In all cases we use $O(n)$ space.*

Acknowledgments

We thank Otfried Cheong, Joachim Gudmundsson, and Marc Pouget for fruitful discussions on early versions of this paper, as well as Jean Cardinal and Stefan Langerman for pointing out useful references.

References

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12:38–56, 1991.

[2] R. Atanassov, P. Bose, M. Couture, A. Maheshwari, P. Morin, M. Paquette, M. Smid, and S. Wuhrer. Algorithms for optimal outlier removal. *J. Discrete Alg.*, to appear.

[3] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22(4):547–567, 1999.

[4] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

[5] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

[6] S. Das, P. P. Goswamib, and S. C. Nandya. Smallest k-point enclosing rectangle and square of arbitrary orientation. *Inform. Proc. Lett.*, 94(6):259–266, 2005.

[7] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11(3):321–350, 1994.

[8] J. W. Jaromczyk and M. Kowaluk. Orientation independent covering of point sets in R^2 with pairs of rectangles or optimal squares. In *Abstracts 12th European Workshop Comput. Geom.*, pages 77–84. Universität Münster, 1996.

[9] M. J. Katz, K. Kedem, and M. Segal. Discrete rectilinear 2-center problems. *Comput. Geom. Theory Appl.*, 15:203–214, 2000.

[10] J. Matoušek, E. Welzl, and M. Sharir. A subexponential bound for linear programming and related problems. *Algorithmica*, 16:365–384, 1996.

[11] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[12] M. Segal and K. Kedem. Enclosing k points in the smallest axis parallel rectangle. *Inform. Process. Lett.*, 65:95–99, 1998.

Identifying Well-Covered Minimal Bounding Rectangles in 2D Point Data

Marc van Kreveld[*] Thijs van Lankveld[*] Remco Veltkamp[*]

Abstract

Laser range imaging is an upcoming data source for many fields of research. Although many methods are proposed for finding surfaces and classifying points, finding the bounds of these surfaces is still a difficult problem. We propose a method for finding, in a point set of size n, a rectangle that contains a predefined subset of the points and none of the other points in $O(n \log n)$ time. Further, there may not be places in the rectangle that are too far from all points in the predefined subset.

1 Introduction

A type of data that has lately seen an increase in use is the laser range scan, also known as light detection and ranging (LiDAR) data. This data consists of three-dimensional points that are scanned from surfaces. The accuracy and resolution of the measurements is high. Currently, many scanners also include a photosensitive sensor that assigns a color value to each point. A more complete description of the properties of LiDAR data is given in [9, 10].

When this data is studied interactively it is easy to form a mental image of the geometry of the scene. The collection of 3D points gives insight into the surfaces that were sampled, and many methods have been proposed to automatically detect these surfaces [12, 14]. One influential group of methods is the methods based on RANSAC [3], which uses a minimal random sample to define a surface and iterates to find the surface containing most points. Another group of methods is based on region growing [14], which initializes a bounded surface at a certain point and tries to expand this while maintaining some constraint like maximal deviation from the surface in location or normal. Finally, Funke, Malamatos, and Ray [4] give an approximation algorithm for finding a connected component with comparable normals in a triangulation. In the last two methods, constraints on local planarity might not hold globally.

Though surfaces can be found in the data, these are usually either unbounded or of an arbitrary and overly complex shape. Computing the correct bounded shapes from these surfaces is far from trivial. Problems arise because the measurements are not regularly distributed over the surfaces or not well suited for detecting edges. In many cases it is possible to determine which points are on a surface, but it is still difficult to find the original boundary of this surface. A similar problem arises when we are not interested in the whole surface, but in smaller sections that should be separated from the surface. An example of this is a known surface of a facade in which we want to find the regions that make up windows or shutters based on their color. Comparable work has been done using LiDAR data when the topology of the original object is known [11].

Within the remainder of this paper, we assume the surface has already been extracted using the method of Schnabel, Wahl, and Klein [12]. This shifts the problem from 3D to 2D. Schnabel's method also filters out the points that are not on the surface. An example of LiDAR data after surface detection is shown in Figure 1. We will assume it is known which points are inside the shape we want to find, for example a window or shutter, and we will call these the blue points. All others will be called red points. This distinction may be made by a so-called superpixel method [2].

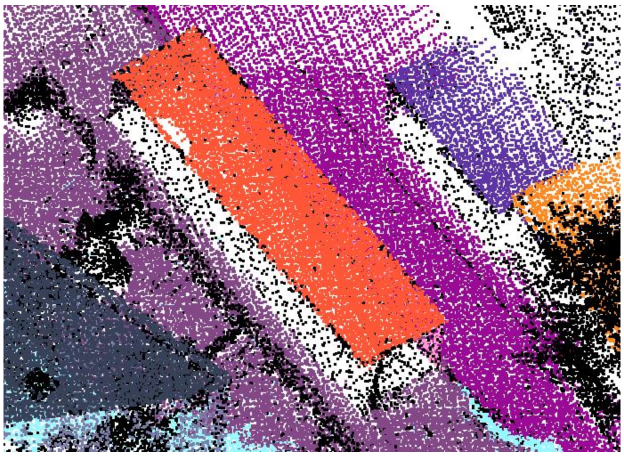


Figure 1: The LiDAR data, in which points close to the same plane have been given the same color. In the rest of the paper, we will call these points blue.

Many objects have rectangular shapes and our method is constrained to finding these. Usually, if there is one rectangle that is valid because it contains all blue points and no red points, then there are many of these rectangles with different orientations

[*]Department of Information and Computing Sciences, Utrecht University, marc@cs.uu.nl, thijsvl@cs.uu.nl, Remco.Veltkamp@cs.uu.nl

and sizes. We constrain the problem to finding the angles $\theta \in [0, \frac{\pi}{2})$ for which there is some *blue* rectangle $\mathcal{R}_\theta$ whose edges are rotated by θ from axis-aligned.

Definition 1 *A rectangle $\mathcal{R}$ is blue with respect to point sets B and R if and only if it contains all points in B and no points in R.*

For our method we use the minimal oriented bounding box at angle θ to check if there is a blue $\mathcal{R}_\theta$. This is valid because to decide if a blue $\mathcal{R}_\theta$ exists, we can test the smallest one.

Generally, we are mostly interested in rectangles that give a good indication of the distribution of a point set. To incorporate this, the problem is extended to finding rectangles that are well covered by the blue points. For a rectangle $\mathcal{R}$ to be well covered, we require that any point inside $\mathcal{R}$ has a blue point within a pre-specified distance δ.

Definition 2 *A rectangle $\mathcal{R}$ is δ-covered by point set B if and only if the union of all disks with radius δ and center $c \in B$ covers $\mathcal{R}$.*

Once the angles θ are identified for which there is a blue rectangle $\mathcal{R}_\theta$ that is δ-covered by B, these rectangles can be searched for the smallest area or diameter rectangle, or for rectangles with a parallel edge within (other) surfaces. However, these applications fall outside the scope of this paper.

Problem The problem solved algorithmically in this paper is as follows. Let R and B be point sets of combined size n, and let δ be a scalar. Determine all angles $\theta \in [0, \frac{\pi}{2})$ for which there is a blue rectangle $\mathcal{R}_\theta$ that is δ-covered by B.

To solve this problem, we assume the data has two properties. Firstly, we assume the data does not contain noise, unlike the original sensor data. Secondly, as stated earlier, we assume it is known which points fall inside and outside the rectangle, that is, which points are blue and which are red. This distinction may be determined beforehand based on the color of each point, or some other property.

Our problem is a red-blue separability problem; see [1, 5, 7, 8] for other papers. Our problem's most distinguishing feature is the extra requirement that the rectangle that is the separator must be δ-covered by the points that lie inside.

The algorithm we present for solving the problem uses ideas of sweep and scan algorithms. The method most closely related is rotating calipers [13], most notably when rotating calipers is used to find the minimal bounding box.

In the next section an algorithm for solving the problem without coverage requirements is given. Section 3 extends this algorithm to solve the complete problem. Finally, we discuss the results and future research.

2 A simple separation algorithm

This section introduces the basic framework for solving the problem of finding a blue rectangle. The approach is to rotate a rectangle $\mathcal{R}$ around point set B and handle important events as they occur. There are two causes for events.

Firstly, because $\mathcal{R}$ is minimal, all of its edges contain at least one point of B, and an event occurs when these points change. These points and the order in which they change can be determined from the boundary of the convex hull $\mathcal{CH}_B$ of B. The events are the same as those used for determining the oriented minimal bounding box with rotating calipers.

Secondly, an event occurs when a red point r enters or exits $\mathcal{R}$. The angle of these events can be determined from the two lines through r, tangent to $\mathcal{CH}_B$. Note that a red point r will never be inside $\mathcal{R}$ if it is too far from $\mathcal{CH}_B$. Let v_1, v_2 be vertices of $\mathcal{CH}_B$ on the tangents through r. Then r can only enter $\mathcal{R}$ if $\angle v_1 r v_2 \geq \frac{\pi}{2}$.

The events are inserted into a queue sorted on angle of rotation at which they occur. After determining which red points are inside $\mathcal{R}$ before rotation, the events are handled in order. The algorithm keeps track of the number of red points inside $\mathcal{R}$ and whenever $\mathcal{R}$ becomes or stops being blue, this information is stored, together with the current angle. When all events have been handled, all angles θ for which $\mathcal{R}_\theta$ is blue are known.

Theorem 1 *For point sets B and R with total size n, all oriented bounding boxes of B that do not contain any point in R can computed in $O(n \log n)$ time.*

Proof. Constructing $\mathcal{CH}_B$ takes $O(n \log n)$ time. This also gives the angles when the points on the edges of $\mathcal{R}$ change. Using a binary search on the vertices of $\mathcal{CH}_B$, all lines through a red point and tangent to $\mathcal{CH}_B$ can be computed in $O(n \log n)$. Each edge of $\mathcal{R}$ will start containing a vertex of $\mathcal{CH}_B$ once and there are two lines tangent to $\mathcal{CH}_B$ through each red point. This means that there are $O(n)$ events. Sorting the events by angle takes $O(n \log n)$ time. During rotation, after each event it is checked if the rectangle has become empty or non-empty. This takes $O(1)$ time per event. □

3 Requiring δ-coverage

The algorithm of the previous section provides an effective way to determine the angles for which a rectangle is blue. We would also like the rectangles not to contain large regions void of blue points. In other

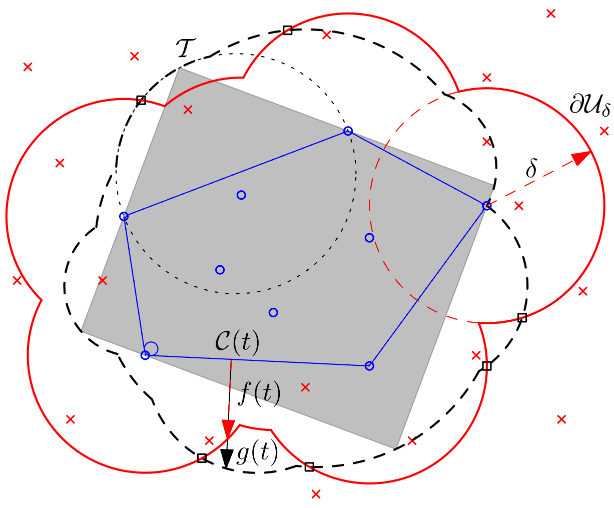


Figure 2: The stuctures used to find the rectangles that contain all blue points B (circles) and no red points R (crosses) when δ-coverage is required. The boundary $\partial\mathcal{U}_\delta$ of the permissible region is shown solid, while the trajectory of the corners $\mathcal{T}$ is shown dashed. The rounded convex hull $\mathcal{C}$ and the functions $f(t)$ and $g(t)$ are also shown.

words, $\mathcal{R}$ must be δ-covered by B for some δ, which we assume is given.

To solve this problem, two additional structures are used: for the permissible region, and for the trajectory of the corners of $\mathcal{R}$. These structures are shown in Figure 2.

The permissible region $\mathcal{U}_\delta$ is the maximal region that is δ-covered by B. It is the union of all δ-disks centered on a point in B. To satisfy the δ-coverage criterion, $\mathcal{R}$ must be completely inside $\mathcal{U}_\delta$. It is well-known that $\mathcal{U}_\delta$ is bounded by $O(n)$ arcs and can be computed in $O(n \log n)$ time. If part of $\mathcal{CH}_B$ is not in $\mathcal{U}_\delta$, then no rectangular separator exists that is δ-covered. In the rest of this paper we will assume $\mathcal{CH}_B \subseteq \mathcal{U}_\delta$ and in this case $\mathcal{U}_\delta$ will have only one connected component whose boundary we denote by $\partial\mathcal{U}_\delta$, see Figure 2.

The trajectory $\mathcal{T}$ is the cycle of circular arcs, shown in Figure 2, that a corner of $\mathcal{R}$ follows as $\mathcal{R}$ is rotated over a full 2π turn. This concept has been previously used by Hoffmann *et al.* [6].

Lemma 2 *$\mathcal{T}$ can be constructed from $\mathcal{CH}_B$ in $O(n)$ time and contains $O(n)$ vertices and circular arcs.*

Proof. Let c be a corner of $\mathcal{R}$ and let $b_1, b_2 \in B$ be the contact points on the two edges of $\mathcal{R}$ incident to c. It follows from Thales' theorem that, while the contact points are unchanged, c follows a circular arc on the circle whose diameter is b_1b_2. A new arc starts only when a contact point changes, and this happens $O(n)$ times. The sequence of the $O(n)$ arcs forms $\mathcal{T}$. □

Because the center of each arc in $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ is on $\mathcal{CH}_B$ and no arc intersects this cycle, the following observation can be made.

Observation 1 *Any outward ray emanating perpendicular to an edge of $\mathcal{CH}_B$ intersects $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ exactly once.*

The algorithm that solves the problem is based on the algorithm given in Section 2. The only difference is the addition of events when $\mathcal{R}$ starts and stops being δ-covered by B. There are two ways in which this can happen. Either during rotation an edge of $\mathcal{R}$ passes over a vertex of $\partial\mathcal{U}_\delta$, or a corner of $\mathcal{R}$ passes over an arc of $\partial\mathcal{U}_\delta$.

The vertices of $\partial\mathcal{U}_\delta$ are handled exactly like extra red points. This will add $O(n)$ events.

The events caused by a corner of $\mathcal{R}$ passing over $\partial\mathcal{U}_\delta$ happen exactly at the intersections of $\partial\mathcal{U}_\delta$ and $\mathcal{T}$, shown as square markers in Figure 2.

Lemma 3 *If $\mathcal{U}_\delta$ covers $\mathcal{CH}_B$, there are $O(n)$ intersections between $\partial\mathcal{U}_\delta$ and $\mathcal{T}$.*

Proof. The proof builds on the property that two piecewise simple functions with n sections have $O(n)$ intersections that can be computed in $O(n)$ time. By simple we mean that two such curves can intersect each other only a constant number of times.

To use this property, we must define two functions that have the same value at some argument if and only if $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ intersect. To create these functions, an extra structure is introduced to parameterize $\partial\mathcal{U}_\delta$ and $\mathcal{T}$, and thereby give the argument to define the functions. We will show that the functions created using this structure "detect" all intersections of $\partial\mathcal{U}_\delta$ and $\mathcal{T}$, which completes the proof.

Let $\mathcal{C}$, shown in Figure 2, be the boundary of the morphological opening, the dilation of the erosion, of the region inside $\mathcal{CH}_B$ using an ϵ-disk. The radius $\epsilon > 0$ is chosen such that for all vertices of $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ that are not on a vertex of $\mathcal{CH}_B$, the closest point on $\mathcal{CH}_B$ is also on $\mathcal{C}$.

Let $\mathcal{C}(t)$ be the point reached by following $\mathcal{C}$ from its topmost point for the fraction t of its length, and let $r(\mathcal{C}(t))$ be the outward ray emanating perpendicularly from $\mathcal{C}(t)$. Now $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ define functions by taking $f(t) = dist(\mathcal{C}(t), r(\mathcal{C}(t)) \cap \partial\mathcal{U}_\delta)$ and $g(t) = dist(\mathcal{C}(t), r(\mathcal{C}(t)) \cap \mathcal{T})$. These functions $f(t)$ and $g(t)$, shown in Figure 2, are well-defined: for each t, $r(\mathcal{C}(t))$ intersects $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ exactly once. This is true for each straight part of $\mathcal{C}$ because of Observation 1. Furthermore, for every circular arc of $\mathcal{C}$, all rays starting in the center of the corresponding ϵ-disk and intersecting the arc, intersect the same arcs of $\partial\mathcal{U}_\delta$ and $\mathcal{T}$; the only exception to this is when a vertex of $\mathcal{T}$ is on a vertex of $\mathcal{CH}_B$. But in this case, such a ray will intersect $\mathcal{T}$ once as well.

The properties of $f(t)$ and $g(t)$ ensure that each point on $\partial\mathcal{U}_\delta$ and $\mathcal{T}$ occurs for some t as the intersection of $r(\mathcal{C}(t))$ and $\partial\mathcal{U}_\delta$, respectively $\mathcal{T}$, and this is the only intersection point for that ray. Hence, $f(t)$ and $g(t)$ have the property that a value of t for which $f(t) = g(t)$ corresponds one-to-one to an intersection point of $\partial\mathcal{U}_\delta$ and $\mathcal{T}$. Since $f(t)$ and $g(t)$ are also piecewise simple, the result follows. $\square$

Lemma 3 implies that there are a linear number of events of the type where a corner of $\mathcal{R}$ is at distance exactly δ from the nearest blue point. These are additional to the ones of the algorithm presented in Section 2 and the vertices of $\partial\mathcal{U}_\delta$. Handling these events is trivial. This leads to the following result.

Theorem 4 *For point sets R and B with total size n, and scalar δ, all angles $\theta \in [0, \frac{\pi}{2})$ for which there is a blue rectangle $\mathcal{R}_\theta$ that is δ-covered by B can be computed in $O(n \log n)$ time.*

4 Discussion

We presented an algorithm that computes all rectangles that tightly cover the 'blue' points in one set but not the 'red' points in another set. An extension of this algorithm also guarantees that no part of the rectangle does not have a blue point nearby. The algorithm is efficient and relatively simple to implement. To make it more useful for the application of geometric model reconstruction from LiDAR data, a number of extensions may be considered.

Firstly, we may need to allow a small number of red points inside the rectangle, because errors may have been made in the classification into red and blue points. It is easy to extend our algorithm to allow up to some pre-specified number of red points inside.

Secondly, we may want to allow a small number of blue points to be outside the rectangle. This extension appears considerably harder, and we do not expect to achieve the same running time in this case.

Thirdly, we may want to allow a small part of the area inside the rectangle to be not δ-covered by the blue points. Due to minor occlusion, it may be that some blue points are missing in a region. This extension is also difficult to incorporate, although an approximation version seems possible.

Fourthly, it would be useful to extend the algorithm to finding different shapes than rectangles, like L-shapes, without complicating the algorithm too much.

A final question is whether our algorithm or some adaptation of it will perform well on data that it would be given in real applications. Only by experimentation can we discover whether the quality of the data (density of sampling, limited presence of noise, proper classification of inliers and outliers) is sufficient to make our approach work well.

Acknowledgments This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

References

[1] P. K. Agarwal, B. Aronov, and V. Koltun. Efficient algorithms for bichromatic separability. *ACM Trans. on Alg.*, 2(2):209–227, 2006.

[2] P. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. of Comp. Vis.*, 59(2), 2004.

[3] M. A. Fischler and R. C. Bolles. RANdom SAmple Consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Graphics and Image Processing*, 24(6):381–395, 1981.

[4] S. Funke, T. Malamatos, and R. Ray. Finding planar regions in a terrain: in practice and with a guarantree. In *SCG '04: Proc. of the 20th Annual Symp. on Comp. Geom.*, pages 96–105, New York, NY, USA, 2004. ACM.

[5] S. Har-Peled and V. Koltun. Separability with outliers. In *ISAAC*, pages 28–39, 2005.

[6] F. Hoffmann, C. Icking, R. Klein, and K. Klaus. The polygon exploration problem II: The angle hull. Technical Report 245, Fernuniversität Hagen, Praktische Informatik VI,, 1998.

[7] F. Hurtado, M. Mora, P. A. Ramos, and C. Seara. Separability by two lines and by nearly straight polygonal chains. *Discrete Applied Mathematics*, 144(1–2):110–122, 2004.

[8] F. Hurtado, C. Seara, and S. Sethia. Red-blue separability problems in 3d. *Int. J. Comp. Geom. Appl.*, 15(2):167–192, 2005.

[9] John Chance Land Surveys and Fugro. Fli-map specifications. http://www.flimap.com/site47.php.

[10] H.-G. Maas. Fast determination of parametric house models from dense airborne laserscanner data. In *ISPRS Workshop on Mobile Technology*, Bangkok, Thailand, April 1999.

[11] S. Pu and G. Vosselman. Automatic extraction of building features from terrestrial laser scanning. *Int. Arch. of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5), September 2006.

[12] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[13] G. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. of the IEEE MELECON '83*, pages A10.02/1–4, 1983.

[14] Y.-H. Tseng, K.-P. Tang, and F.-C. Chou. *Surface reconstruction from LiDAR data with extended snake theory*, volume 4679 of *Lecture Notes in Computer Science*, pages 479–492. Springer Berlin / Heidelberg, August 2007.

Fixed-parameter tractability and lower bounds for stabbing problems

Panos Giannopoulos*† Christian Knauer*† Günter Rote* Daniel Werner*

Abstract

In this paper we study the parameterized complexity of several "stabbing" problems.

For a general class of objects in $\mathbb{R}^2$, we show that the (decision) problem of stabbing a set of translates of such an object with k axis-parallel lines is W[1]-hard with respect to k, and thus, not fixed-parameter tractable, unless W[1]=FPT. When the lines can have arbitrary directions, it is even W[1]-hard for disjoint objects. Nevertheless, for some special cases such as stabbing disjoint unit squares with axis-parallel lines, we show that the problem is fixed-parameter tractable by giving an algorithm that runs in $\mathcal{O}(n \log n)$ time for every fixed k.

We also show that deciding whether n given unit balls in $\mathbb{R}^d$ can be stabbed by one line (the decision version of the minimum enclosing cylinder problem) is W[1]-hard with respect to the dimension d.

1 Introduction

We study several instances of the following general stabbing problem: Given a set S of n translates of an object in $\mathbb{R}^d$ (e.g., squares or circles in the plane), find a set of k lines with the property that every object in S is "stabbed" (intersected) by at least one line. All these problems are known to be NP-hard and for most of them only polynomial time constant-factor approximation algorithms are known. We study these problems from a *parameterized complexity* point of view: We examine whether algorithms that run in $\mathcal{O}(f(k,d) \cdot n^c)$ time on inputs of size n (where c is a constant independent of k, d, n) do exist; see [2] for an introduction to parameterized complexity theory.

Results. For a general class of objects in $\mathbb{R}^2$, we show that the problem of stabbing with (even axis-parallel) lines is W[1]-hard with respect to k, and thus, not fixed-parameter tractable, unless W[1]=FPT.

Theorem 1 *Let O be a connected object in the plane (which is not a point). (i) The problem of stabbing a set of translates of O is W[1]–hard with respect to the parameter k if the stabbing lines are to be parallel to two different directions u, v, unless O is contained in a line parallel to u or v. (ii) Stabbing disjoint translates of O with k lines in arbitrary directions is W[1]–hard with respect to the parameter k.*

Corollary 2 *Stabbing a set of disjoint rectangles in the plane by k axis-parallel lines is W[1]-hard with respect to k.*

Let $\mathcal{D}$ be a set of directions. A line with a direction from $\mathcal{D}$ is called a $\mathcal{D}$-line. A set of objects with the property that the maximum number of objects that can be simultaneously intersected by two $\mathcal{D}$-lines with different directions is bounded by $c \in \mathbb{N}$ is called *c–shallow for $\mathcal{D}$*.

Theorem 3 *Stabbing sets that are $\mathcal{O}(1)$-shallow for $\mathcal{D}$ with k $\mathcal{D}$-lines can be decided in $\mathcal{O}(n \log n)$ time for every fixed k.*

Finally, we prove that the minimum enclosing cylinder of n points in $\mathbb{R}^d$ can probably not be computed by an algorithm with a running time of the form $\mathcal{O}(f(d) \cdot n^c)$, by showing the corresponding decision problem to be W[1]-hard:

Theorem 4 *Stabbing unit balls in $\mathbb{R}^d$ with a line is W[1]-hard with respect to the parameter d.*

Our reduction also implies that for this problem and, consequently, for the problem of computing the minimum enclosing cylinder of n points in $\mathbb{R}^d$, no $n^{o(d)}$-time algorithm exists, unless the Exponential Time Hypothesis (which asserts that n-variable 3SAT cannot be solved in $2^{o(n)}$-time) fails to hold [4].

Related Results. Langerman and Morin [5] showed that an abstract NP-hard covering problem that models a number of concrete geometric (as well as purely combinatorial) covering problems is in FPT.

Stabbing unit balls in unbounded dimension was shown to be NP-hard by Megiddo [6]. Hasin and Megiddo [3] developed constant factor approximations for special instances of the stabbing problem.

In a recent work, and independently of our results, Dom et al. [1] also show that the problem of stabbing (overlapping) rectangles with axis-parallel lines

*Institut für Informatik, Freie Universität Berlin, Takusstraße 9, D-14195 Berlin, Germany, {panos, knauer, rote, dwerner}@inf.fu-berlin.de.

†This research was supported by the German Science Foundation (DFG) under grant Kn 591/3-1.

is W[1]-hard (by a different and more complicated reduction from the multi-colored k-clique problem) but in FPT for disjoint unit squares.

2 Stabbing with k lines

2.1 Hardness Results

We first prove that stabbing axis-parallel unit squares with k axis-parallel lines is W[1]-hard with respect to k by an fpt-reduction from the W[1]-complete k-clique problem in directed graphs (c.f. [2]). For proving Theorem 1, we modify our construction in order to handle general objects and lines of arbitrary direction. In this section we consider all objects to be open.

Let $[n] = \{1, \ldots, n\}$ and $G = ([n], E)$ be a simple directed graph with no self edges. From G we construct a set $\mathcal{S}$ of axis-parallel squares in $\mathbb{R}^2$ such that $\mathcal{S}$ can be stabbed by $6k$ axis-parallel lines if and only if G has a clique of size k.

We first create instances $\mathcal{S}^*(G, k)$ with squares of *two different* sizes. There will be k horizontal and k vertical "double strips" $S_h^1, \ldots, S_h^k$ and $S_v^1, \ldots, S_v^k$, respectively, to choose lines from (c.f. Fig. 1). Out of each of those strips, two "consistent" lines will have to be chosen in order to get a solution of the specified size. Around every intersection of a vertical and a horizontal double strip, we place a gadget consisting of a set of squares within a region of suitable size. The gadgets represent the adjacency relation of the graph. We will ensure that (**P0**): Any selection of $2k$ such line pairs corresponds to a set C of vertices of G. (**P1**): Two orthogonal line pairs in strips S_h^i, S_v^j with $i \neq j$ will intersect all the squares in these strips iff these line pairs correspond to vertices that are connected in G. (**P2**): Two orthogonal line pairs in strips S_h^i, S_v^i will intersect all the squares in these strips iff these line pairs correspond to the same vertex. The $2k$ line pairs intersect all the squares iff C forms a k–clique. We will need $2k$ more lines to guarantee the consistency of such a selection.

Let $\square_l(x, y)$ denote the axis-parallel square with side length l and lower left corner (x, y). A gadget T will consist of a collection of axis-parallel squares. $T(x, y)$ denotes a copy of T whose squares are placed relative to (x, y).

The A–gadget (adjacency). A-gadgets represent the adjacency relation of the graph G. Each consists of the set $A := \{\square_{n-1}(i, j) \mid (i, j) \notin E\}$. There are $2n$ *v(ertex)–strips* strips in each direction to chose (combinatorially) different axis-parallel lines through an A–gadget from. If a line l goes through the i-th v–strip of the gadget we say that it *represents* the vertex $\mathrm{rep}(l) := i \mod n$ of G. If $0 < i < n$ the line l is called *negative*, otherwise it is called *positive*. Two lines are called *antipodal* if one is positive and the other is negative.

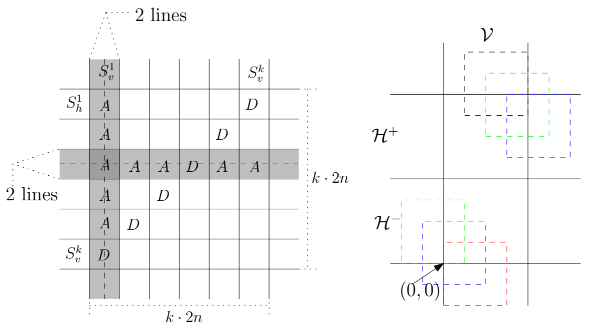


Figure 1: Double strips and a C_h–gadget ($n = 4$).

The construction ensures that if we have two antipodal horizontal lines that represent vertex i and two antipodal vertical lines that represent vertex j, then they intersect all the squares inside A iff i and j are adjacent in G, which will ensure (P1).

The D–gadget (diagonal). This type of gadget is an A–gadget for the graph with adjacency defined by the identity,i.e., $D := \{\square_{n-1}(i, j) \mid 1 \leq i \neq j \leq n\}$. The only line pairs allowed are those that correspond to the same vertex, which will ensure property (P2).

The F–gadget (forcing). This type of gadget will be used to force the existence of horizontal (or vertical) lines in a specified strip in any solution of size $6k$. We define them as $F_h := \{\square_n(-i \cdot n, 0) \mid 1 \leq i \leq 6k + 1\}$ and $F_v := \{\square_n(0, -i \cdot n) \mid 1 \leq i \leq 6k + 1\}$.

To intersect all squares of an F_h–gadget with only vertical lines requires at least $6k + 1$ such lines, thus in any solution with $\leq 6k$ lines there must be a horizontal line intersecting the gadget.

The C–gadget (consistency). This type of gadget is used to guarantee a certain distance between two lines *of the same direction* corresponding to the same double–strip of an A–(or D)–gadget. With this it will be possible to identify such line pairs with the vertices (P0). It will consist of three F–gadgets and $2(n - 1)$ additional squares. We describe the C_h–gadget for horizontal lines only (C_v–gadgets are defined analogously). C_h consists of the union of the two sets $\{\square_{n-1}(1 - i, i - n + 1) \mid 1 \leq i \leq n - 1\}$ and $\{\square_{n-1}(n + 1 - i, n + i - 1) \mid 2 \leq i \leq n\}$ together with three F–gadgets that force the existence of lines in the regions $\mathcal{H}^- = \mathbb{R} \times (0, n)$, $\mathcal{H}^+ = \mathbb{R} \times (n, 2n)$ and $\mathcal{V} = (0, n) \times \mathbb{R}$. See Figure 1 for an example.

The next lemma states the main property of the C_h–gadgets (C_v–gadgets have a symmetric property):

Lemma 5 *Given two antipodal horizontal lines h^-, h^+, in $\mathcal{H}^-, \mathcal{H}^+$, respectively, then there exists a vertical line that together with the others intersects all of the squares belonging to a C_h–gadget iff $\mathrm{rep}(h^+) \geq \mathrm{rep}(h^-)$. In fact, we can always assume that $\mathrm{rep}(h^+) = \mathrm{rep}(h^-)$.*

Putting it all together. We now describe the placement of the gadgets defined above. The main part, expressing the adjacency–relation of the graph, will be a $k \times k$ grid of A–gadgets and D–gadgets: $\mathcal{A} := \{A(i \cdot 3n, j \cdot 3n) \mid 1 \le i \ne j \le k\}$ and $\mathcal{D} := \{D(i \cdot 3n, i \cdot 3n) \mid 1 \le i \le k\}$. Around this grid, we add several C–gadgets to allow only specific solutions: $\mathcal{C}_h := \{C_h(-i \cdot 3n, i \cdot 3n) \mid 1 \le i \le k\}$ and $\mathcal{C}_v := \{C_v(i \cdot 3n, -i \cdot 3n) \mid 1 \le i \le k\}$. The size of the set $\mathcal{S}^*(G,k) := \mathcal{A} \cup \mathcal{D} \cup \mathcal{C}_h \cup \mathcal{C}_v$ is $\mathcal{O}\left(k^2n^2\right)$.

Lemma 6 *$\mathcal{S}^*(G,k)$ can be stabbed by $6k$ axis-parallel lines iff G has a k–clique.*

To yield set $\mathcal{S}(G,k)$ of unit squares, we shrink the squares in the F–gadgets by letting $F_h := \{\square_{n-1}(-in, 1/2) \mid 1 \le i \le 6k+1\}$ (similarly for F_v).

Corollary 2 follows by shifting all the squares by a small amount and replacing all (parallel) diagonals by very thin pairwise disjoint rectangles.

Arbitrary directions and general objects. First, the forcing gadgets are modified to contain n^2 squares; this will ensure that the chosen lines have to be "almost" parallel. Then, the squares are shrinked by some small value ϵ; this ensures that for each almost parallel line there is a parallel line that intersects the same squares. Thus, in any solution the lines can actually be assumed to be axis-parallel, i.e., Lemma 6 holds also for this new set. Again, shifting the squares a little and replacing the diagonals by rectangles as above yields a set of disjoint rectangles. This set is then linearly transformed to yield a combinatorially equivalent set of disjoint unit squares.

To prove the analogous results for arbitrary, connected objects we simply linearly transform them so that their axis-parallel bounding box is a unit square.

2.2 Fixed Parameter Tractable Cases

We prove that stabbing *disjoint* axis-parallel unit squares in the plane with k axis-parallel lines is fpt (with respect to k) by combining data reduction with branching and kernelization techniques. The algorithm can be easily modified to handle the case of general objects and line directions for Theorem 3. In this section we consider all sets to be closed.

Let $\mathcal{S}$ be a set of n disjoint axis-parallel unit squares. We want to determine if $\mathcal{S}$ can be stabbed by k axis-parallel lines. It suffices to consider only lines that support the boundary of a square in $\mathcal{S}$; there are at most $4n$ of them. For a line l let $I(l)$ denote the set of squares that are stabbed by l. We first apply the following data reduction rule to $\mathcal{S}$:

DR: *For all $\kappa > k+1$ squares with the same x–coordinates, delete all but $k+1$ of them, and also for $\kappa > k+1$ squares with the same y–coordinates.*

A set on which DR has been applied is called a *DR–set.* The algorithm is based on the following lemma.

Lemma 7 *Let l be a horizontal line that intersects $\kappa > k$ unit squares $I(l) = \{\square(x_i, y_i) \mid 1 \le i \le \kappa\} \subseteq \mathcal{S}$. Then in order to stab the set $\mathcal{S}$ with k lines, there has to be a horizontal line l^* that intersects at least two squares from $I(l)$. l^* can be chosen from $B(I(l)) := \{y = y_i \mid 1 \le i \le \kappa\} \cup \{y = y_i + 1 \mid 1 \le i \le \kappa\}$.*

An analogous lemma holds for vertical lines as well. Observe that for a DR–set $\mathcal{S}$, if there is an axis-parallel line l with $|I(l)| > 2k+1$, then there is also a line l^* parallel to l with $k+1 \le |I(l^*)| \le 2k+1$. For such a line we use Lemma 7 by branching on all the $|B(I(l))|$ possible lines. If no such line exists, we end up with a problem kernel (in each branch): All the lines now intersect at most k squares, so the SOLVE function simply rejects if the number of squares left is more than k^2 and otherwise uses brute force.

```
STAB($S$, $k$)
  if $S = \emptyset$ "SOLUTION FOUND"
  else if $k = 0$ return
  apply DR
  if there is a line $l$ with $k+1 \le |I(l)| \le 2k+1$
      for all lines $l'$ from the set $B(I(l))$
          STAB($S - I(l')$, $k-1$)
  else SOLVE($S$, $k$)
```

Lemma 8 *If there is a solution of size k, the above algorithm finds a solution of size $k' \le k$.*

The algorithm branches on at most $2(2k+1)$ possibilities at most k times and each call of the STAB procedure needs $\mathcal{O}(n \log n)$ steps. In each branch it ends up with a problem kernel, which can be solved in time $\left(4k^2\right)^k$ by exhaustive search, so the total running time is $\mathcal{O}(n \log n)$ for every fixed k.

3 Stabbing balls with one line

We show that the problem of stabbing unit balls in $\mathbb{R}^d$ with a line is W[1]-hard with respect to d by an fpt-reduction from the W[1]-complete k-independent set problem is general graphs [2]. Given an undirected graph $G([n], E)$ we construct a set $\mathcal{B}$ of balls in $\mathbb{R}^{2k}$ such that $\mathcal{B}$ can be stabbed by a line if and only if G has an independent set of size k.

For every ball $B \in \mathcal{B}$ we will also have $-B \in \mathcal{B}$. This allows us to restrict our attention to lines through the origin. For a line l, let $\vec{l}$ be its unit direction vector. We view $\mathbb{R}^{2k}$ as the product of k orthogonal planes $E_1, \ldots, E_k$, where each E_i has coordinate axes X_i, Y_i. The component on X_i, Y_i of a point p is denoted by $x_i(p)$, $y_i(p)$ respectively.

For each plane E_i, we define $2n$ $2k$-dimensional balls, with centers regularly spaced on the unit circle

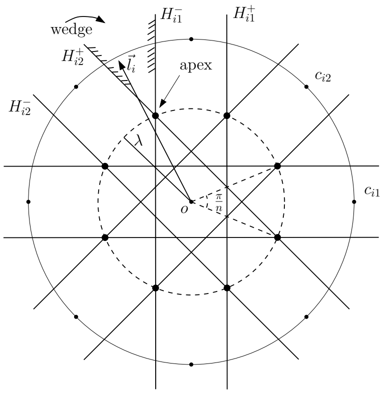


Figure 2: Centers of the balls and their respective half-planes and wedges on a plane E_i, for $n = 4$.

on E_i centered at the origin o. Let $c_{iu} \in E_i$ be the center of the ball B_{iu}, $u \in [2n]$, with $x_i(c_{iu}) = \cos(u-1)\frac{\pi}{n}$ and $y_i(c_{iu}) = \sin(u-1)\frac{\pi}{n}$. We define a scaffolding ball set $\mathcal{B}^0 = \{B_{iu}, i = 1, \ldots, k \text{ and } u = 1, \ldots, 2n\}$. We have $|\mathcal{B}^0| = 2nk$. All balls in $\mathcal{B}^0$ will have the same radius $r < 1$, to be defined later.

Two antipodal balls B, $-B$ are stabbed by the same set of lines. A line l stabs $\mathcal{B}^0$ if and only if it satisfies the system of nk inequalities: $(c_{iu} \cdot \vec{l})^2 \geq ||c_{iu}||^2 - r^2$, for $i = 1, \ldots, k$ and $u = 1, \ldots, n$.

Geometrically, the inequality asserting that l stabs B_{iu} says that the projection $\vec{l_i}$ of $\vec{l}$ on the plane E_i lies in one of the half-planes $H_{iu}^+ = \{p \in E_i | c_{iu} \cdot p \geq \sqrt{||c_{iu}||^2 - r^2}\}$ or $H_{iu}^- = \{p \in E_i | c_{iu} \cdot p \leq -\sqrt{||c_{iu}||^2 - r^2}\}$. Consider the situation on a plane E_i. Line l stabs all balls B_{iu} (centered on E_i) if and and only if $\vec{l_i}$ lies in one of the $2n$ wedges $\pm(H_{i1}^- \cap H_{i2}^+), \ldots, \pm(H_{i(n-1)}^- \cap H_{in}^+)$, and $\pm(H_{i1}^- \cap H_{in}^-)$; see Fig 2. The apexes of the wedges are regularly spaced on a circle of some radius λ. By choosing $r = \sqrt{1 - (1 - \cos\frac{\pi}{n})/(2k)}$ we have that $\lambda = 1/(\sqrt{k})$.

Since $\vec{l} \in \mathbb{R}^{2k}$ is unit, we have that $||\vec{l_i}|| = 1/(\sqrt{k})$, for every $i \in \{1, \ldots, k\}$. Hence, for line l to stab all balls in $\mathcal{B}^0$, every projection $\vec{l_i}$ must be one of the apexes in A_i. There are $2n$ choices for each $\vec{l_i}$, and the total number of lines that stab $\mathcal{B}^0$ is $n^k 2^{k-1}$.

For a tuple $(u_1, \ldots, u_k) \in [2n]^k$, we denote by $l(u_1, \ldots, u_k)$ the line with vector $\frac{1}{\sqrt{k}}(\cos(2u_1 - 1)\frac{\pi}{2n}, \sin(2u_1 - 1)\frac{\pi}{2n}, \ldots, \cos(2u_k - 1)\frac{\pi}{2n}, \sin(2u_k - 1)\frac{\pi}{2n})$. Two lines $l(u_1, u_2, ..., u_k)$ and $l(v_1, v_2, ..., v_k)$ are said to be equivalent if $u_i \equiv v_i(\mod n)$, for all i. We have n^k equivalence classes $L(u_1, \ldots, u_k)$, with $(u_1, \ldots, u_k) \in [n]^k$, where each class consists of 2^{k-1} lines. There is a bijection between the possible equivalence classes of lines that stab $\mathcal{B}^0$ and $[n]^k$.

Constraint balls. For each pair of distinct indices $i \neq j$ $(1 \leq i, j \leq k)$ and for each pair of (possibly equal) vertices $u, v \in [n]$, we define a constraint set of balls $\mathcal{B}_{ij}^{uv}$ with the property that (all lines in) all classes $L(u_1, \ldots, u_k)$ stab $\mathcal{B}_{ij}^{uv}$ except those with $u_i = u$ and $u_j = v$. The centers of the balls in $\mathcal{B}_{ij}^{uv}$ lie in the 4-space $E_i \times E_j$. All lines in a class $L(u_1, \ldots, u_k)$ project into only two lines on $E_i \times E_j$. We use a ball B_{ij}^{uv} (to be defined shortly) that is stabbed by *all* lines $l(u_1, \ldots, u_k)$ except those with $u_i = u$ and $u_j = v$. Similarly, we use a ball $B_{ij}^{u\bar{v}}$ that is stabbed by *all* lines $l(u_1, \ldots, u_k)$ except those with $u_i = u$ and $u_j = \bar{v}$, where $\bar{v} = v + n$. We have, $\mathcal{B}_{ij}^{uv} = \{\pm B_{ij}^{uv}, \pm B_{ij}^{u\bar{v}}\}$.

Consider a line $l = l(u_1, \ldots, u_k)$ with $u_i = u$ and $u_j = v$. The center c_{ij}^{uv} of B_{ij}^{uv} will lie on a direction $\vec{z} \in E_i \times E_j$ orthogonal to $\vec{l}$, but for which $\vec{l'} \cdot \vec{z} \neq 0$ for any line $l' = l(u_1, \ldots, u_k)$ with $u_i \neq u$ or $u_j \neq v$. Let ω be the angle between $\vec{l'}$ and $\vec{z}$. We choose $\vec{z}$ such that $|\cos\omega| > \frac{\lambda}{\sqrt{k}}$, where $\lambda < 1$ is an appropriate function of n. This helps us place B_{ij}^{uv} sufficiently close to the origin so that it is still intersected by l'. We can choose any point c_{ij}^{uv} on z with $r < ||c_{ij}^{uv}|| < r\sqrt{k/(k - \lambda^2)}$.

We add to $\mathcal{B}^0$ the $4n\binom{k}{2}$ balls in $\mathcal{B}_V = \bigcup \mathcal{B}_{ij}^{uu}$, $1 \leq u \leq n$, $1 \leq i < j \leq k$, to ensure that all components u_i in a solution (class of lines $L(u_1, \ldots, u_k)$) are distinct. For each edge $uv \in E$ we also add the balls in $k(k-1)$ sets $\mathcal{B}_{ij}^{uv}$, with $i \neq j$. This ensures that the remaining classes of lines $L(u_1, \ldots, u_k)$ represent independent sets of size k. In total, the edges are represented by $\mathcal{B}_E = \bigcup \mathcal{B}_{ij}^{uv}$, $uv \in E$, $1 \leq i, j \leq k$, $i \neq j$. The final set $\mathcal{B} = \mathcal{B}^0 \cup \mathcal{B}_V \cup \mathcal{B}_E$ has $2nk + 4\binom{k}{2}(n + 2|E|)$ balls. The constraint sets of balls exclude tuples with two equal indices $u_i = u_j$ or with indices u_i, u_j when $u_i u_j \in E$, thus, the classes of lines that stab B represent exactly the independent sets of G.

Lemma 9 *Set $\mathcal{B}$ can be stabbed by a line if an only if G has an independent set of size k.*

References

[1] M. Dom, M. R. Fellows, and F. A. Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. In *Proc. of the 3rd WALCOM*, LNCS, 2009. To appear.

[2] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[3] R. Hassin and N. Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30:29–42, 1991.

[4] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[5] S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

[6] N. Megiddo. On the complexity of some geometric problems in unbounded dimension. *J. Symb. Comput*, 10:327–334, 1990.

Robust Voronoi-based Curvature and Feature Estimation

Quentin Mérigot* Maks Ovsjanikov† Leonidas Guibas‡

Abstract

In this paper we present a method for extracting principal curvature directions from a point cloud sampling of a piecewise-smooth surface, using convolved covariance matrices of Voronoi cells of the point cloud. The same method can be used to extract the location and direction of sharp edges. The convergence and correctness properties of this method are analyzed both theoretically and experimentally, under various noise and sampling conditions.

1 Introduction

Curvature and feature estimation have many applications in graphics and geometry processing. Principal curvatures are rotation-invariant local descriptors, which together with principal curvature directions have proven useful in symmetry detection, global matching, modeling and rendering of point-based surfaces, among others. The location of sharp edges and highly curved areas of the surface is a precious piece of information in settings that include feature-aware reconstruction or remeshing, clustering and non photorealistic rendering.

Prior work. Recent work on curvature estimation on *meshes* includes [5], with which we share a measure-theoretic approach. Our work is also inspired by Voronoi-based normal estimators such as [4] and [1].

Contributions. Given a scale parameter R, we associate to *every* compact set $K \subseteq \mathbb{R}^d$ a tensor-valued measure $\mathcal{V}_{K,R}$ encoding the covariance of the normal cones, which we call the *Voronoi Covariance Measure* of K. This object, which is a tensor-valued version of the *boundary measure* introduced in [3], is a formalized and generalized version of the covariance matrices of Voronoi cells used for normal estimation in [1].

If K is a point cloud, then $\mathcal{V}_{K,R}$ corresponds to the covariance matrices of the Voronoi cells of K intersected with an offset of radius R. Convolving the VCM with a kernel χ then corresponds to simply taking a weighted sum of the covariance matrices in a neighborhood.

We prove that if S is a smooth surface, and χ_r is the indicator function of a ball of radius r centered at the origin, the three eigenvectors of the convolved VCM $\mathcal{V}_{S,R} * \chi_r(p)$ with $p \in S$ are close to the normal and the two principal curvature direction at p, provided that R and r are chosen small enough. Similarly, if p belongs to a sharp edge of a piecewise surface, the eigenvector corresponding to the smallest eigenvalue of $\mathcal{V}_{S,R} * \chi_r(p)$ is close to the tangent direction to that sharp feature.

We then provide a very general stability result for VCMs, proving that if the symmetric Hausdorff distance between K and K' is less than ε, then for every $p \in \mathbb{R}^d$, $\|\mathcal{V}_{K,R} * \chi_r(p) - \mathcal{V}_{K',R} * \chi_r(p)\|_{\mathrm{op}} = O(\sqrt{\varepsilon})$, where $\|.\|_{\mathrm{op}}$ is the usual operator norm. As a consequence, if C is a good point cloud approximation of a piecewise smooth surface S, and $p \in S$, then the eigenvectors of $\mathcal{V}_{C,R}(p)$ will be a good approximation of the eigenvectors of $\mathcal{V}_{S,R}(p)$. Note that since the Hausdorff distance is a purely geometric notion, our result does not make any assumption on the nature of the sampling, such as uniform density of the sample points, and handles strongly anisotropic sampling gracefully.

Finally, we present two algorithms for computing the VCM of a point cloud: an easy-to-implement Monte-Carlo algorithm, and a faster algorithm that requires tessellating the intersection of a ball with a Voronoi cell.

2 The Voronoi Covariance Measure

Mathematical background. If $K \subseteq \mathbb{R}^d$ is a compact set, we will denote by d_K the *distance function* to K, *i.e.* for any $x \in \mathbb{R}^d$, $\mathrm{d}_K(x) = \min\{\mathrm{d}(x, y); y \in K\}$. The *R-offset* of K is the set of points at distance at most R of K; we will denote it by K^R.

The projection of x on K is the set of points $y \in K$ which realize this minimum; the set of points for which several projections exist is the medial axis $\mathcal{M}(K)$ of K. The distance of K to its medial axis is called the *minimum local feature size*, or *reach* of K.

The *projection function* maps a point in $\mathbb{R}^d \setminus \mathcal{M}(K)$ to its only closest point $\mathrm{p}_K(x)$ in $K \subseteq \mathbb{R}^d$. Since the medial axis has zero d-volume, the projection function is defined almost everywhere. Being only interested in integral properties, we will often abusively consider p_K as defined on the whole space $\mathbb{R}^d$.

Definition 1 *The Voronoi covariance measure $\mathcal{V}_{K,R}$ of a compact $K \subseteq \mathbb{R}^d$ at scale $R > 0$ is a tensor-valued*

*INRIA Sophia-Antipolis, quentin.merigot@sophia.inria.fr
†Stanford University, maks@stanford.edu
‡Stanford University, guibas@cs.stanford.edu

measure on $\mathbb{R}^d$, i.e. it maps any subset $B \subseteq \mathbb{R}^d$ to the following covariance matrix:

$$\mathcal{V}_{K,R}(B) = \int_{\mathrm{p}_K^{-1}(B \cap K) \cap K^R} (x - \mathrm{p}_K(x))(x - \mathrm{p}_K(x))^{\mathbf{T}} \mathrm{d}x$$

The VCM $\mathcal{V}_{K,R}$ can be convolved by a compactly supported function χ on $\mathbb{R}^d$, turning it into the tensor-valued function $\mathcal{V}_{K,R} * \chi$ defined by:

$$\mathcal{V}_{K,R}*\chi(p) = \int_{K^R} (x - \mathrm{p}_K(x))(x - \mathrm{p}_K(x))^{\mathbf{T}} \chi(\mathrm{p}_K(x) - p) \mathrm{d}x$$

The main idea behind these definitions is if p is a point of the compact set K, the set $\mathrm{p}_K^{-1}(\{p\}) \cap K^R$ intuitively corresponds to the normal cone to K at that point, at a scale given by $R > 0$. Indeed, if S is a smooth hypersurface, and $R < \mathrm{reach}(S)$, $\mathrm{p}_S^{-1}(\{p\}) \cap S^R$ is just the segment $[p - R\vec{n}, p + R\vec{n}]$ where $\vec{n}$ is a unit normal at p. If however P is a convex polyhedron, $\mathrm{p}_S^{-1}(\{p\}) \cap S^R$ coincides with the usual normal cone of convex geometry: it is a $(d-k)$-dimensional subset of $\mathbb{R}^d$ if p is on a k-facet of the polyhedron.

When the dimension of the set $\mathrm{p}_K^{-1}(\{p\})$ is less than d, $\mathcal{V}_{K,R}(\{p\})$ is just the zero matrix. Thus, in order to get meaningful information from the VCM, we need to evaluate it on a higher dimensional subset, *e.g.* a small ball $\mathrm{B}(p, r)$ around p. Then, $\mathcal{V}_{K,R}(\mathrm{B}(p, r))$ corresponds to the sum of the (infinitesimal) covariance matrices of the sets $\mathrm{p}_K^{-1}(\{p'\})$, with $p' \in \mathrm{B}(p, r)$. Therefore, we can expect that the eigenvalue analysis of $\mathcal{V}_{K,R}(\mathrm{B}(p, r))$, where $\mathrm{B}(p, r)$ is a small ball around p, will provide information's about the shape and dimension of the normal cone to K around p.

The function $p \mapsto \mathcal{V}_{K,R}(\mathrm{B}(p, r))$ is actually a special case of convolved VCM: if χ_r denotes the indicator function of the ball $\mathrm{B}(0, r)$, it follows at once that $\mathcal{V}_{K,R}(\mathrm{B}(p, r)) = \mathcal{V}_{K,R} * \chi_r(p)$.

VCM of a Point Cloud. If $C = \{p_1, \ldots, p_n\}$ is a point cloud, the projection on C is the function which maps a point x to the center of the Voronoi cell x belongs to, *i.e.* $\mathrm{p}_C^{-1}(p_i) = \mathrm{Vor}(p_i)$. This means that if $B \subseteq \mathbb{R}^d$,

$$\mathcal{V}_{C,R}(B) = \sum_{p_i \in B} \mathrm{cov}(\mathrm{Vor}(p_i) \cap \mathrm{B}(p_i, R), p_i)$$

where $\mathrm{cov}(E, p) = \int_{x \in E} (x - p)(x - p)^{\mathbf{T}} \mathrm{d}x$ is the covariance matrix of the domain E centered at p.

If χ_r is the indicator function of $\mathrm{B}(0, r)$, it follows from the remarks above that:

$$\mathcal{V}_{C,R} * \chi_r(p) = \sum_{p_i \in \mathrm{B}(p,r)} \mathrm{cov}(\mathrm{Vor}(p_i) \cap \mathrm{B}(p_i, R), p_i)$$

VCM of a smooth hypersurface. If S is an orientable smooth surface embedded in $\mathbb{R}^3$, $\vec{n}$ a normal vector field on S and $R_S = \mathrm{reach}(S) > R$, then for every $B \subseteq S$ the VCM evaluated on B is equal (up to a constant) to the covariance matrix of the normals to S whose base point lie in B. More precisely,

$$\mathcal{V}_{S,R}(B) = \frac{2}{3} R^3 [1 + O(R^2/R_S^2)] \int_{p \in B \cap S} \vec{n}(p)\vec{n}(p)^{\mathbf{T}} \mathrm{d}p$$

The proof of this fact is a simple application of the change of variable formula, remarking that $S^R \cap \mathrm{p}_S^{-1}(B)$ is the one-to-one image of the map $\varphi : B \cap S \times [-R, R] \to S^R$, $(p, t) \mapsto p + t\vec{n}(p)$.

If one evaluates $\mathcal{V}_{S,R}$ on a ball $B = \mathrm{B}(p, r)$, where r is a small convolution radius, the resulting covariance matrix will reflect the variation of the normals to S around p, which is known to be related to the second fundamental form and hence curvature.

Precisely, if $\kappa_1(p), \kappa_2(p)$ denote the two principal curvatures at p and $\vec{P}_1(p), \vec{P}_2(p)$ are the corresponding principal curvature directions, one has the following Taylor expansion:

$$\mathcal{V}_{S,E}(B(p, r)) \simeq \frac{2\pi}{3} R^3 r^2 \left[\vec{\mathrm{n}}(p)\vec{\mathrm{n}}(p)^{\mathbf{T}} + \frac{r^2}{4} \sum_{i=1}^{2} \kappa_i^2(p) \vec{P}_i(p) \vec{P}_i(p)^{\mathbf{T}} \right]$$

This proves that the eigenvalues of $\mathcal{V}_{S,E} * \chi_r(p)$ will be (up to a constant) close to 1, $\kappa_1(p)r/2$ and $\kappa_2(p)r/2$, with corresponding eigenvectors $\vec{\mathrm{n}}(p)$, $\vec{P}_1(p)$ and $\vec{P}_1(p)$.

VCM near a sharp edge. Let now S be an embedded piecewise smooth surface. For $p \in S$, the normal cone to S at p, denoted by $\mathcal{N}_p S$, is the positive cone generated by $\{x - p\,;\, \mathrm{p}_K(x) = p\}$. The shape and dimension of this cone is related to the sharpness of S at p, as suggested in Figure 2.1.

Suppose that p lies on a sharp edge of S, and let $\vec{\mathrm{t}}$ be the tangent to this edge at p, and $\vec{\mathrm{u}}$, $\vec{\mathrm{v}}$ be the projections of the two outward normals to S at p on the orthogonal plane to $\vec{\mathrm{t}}$. Suppose moreover that for all $v \in \mathcal{N}_p S$ with $\|v\| \leqslant R$, then $\mathrm{p}_S(p + v) = p$ – this assumption can be seen as a one-sided lower bound on the local feature size. Then,

$$\mathcal{V}_{S,S^R}(\mathrm{B}(p_0, r)) = \frac{R^4 r}{8} [(\alpha - \sin(\alpha))\, e_1 e_1^{\mathbf{T}} + (\alpha + \sin(\alpha))\, e_2 e_2^{\mathbf{T}} + O(r/R)]$$

where $\vec{e_1} = \frac{\vec{\mathrm{u}} - \vec{\mathrm{v}}}{\|\vec{\mathrm{u}} - \vec{\mathrm{v}}\|}$, $\vec{e_2} = \frac{\vec{\mathrm{u}} + \vec{\mathrm{v}}}{\|\vec{\mathrm{u}} + \vec{\mathrm{v}}\|}$, and $\cos(\alpha) = \langle \vec{\mathrm{u}} | \vec{\mathrm{v}} \rangle$.

In particular, the eigendirection corresponding to the smallest eigenvalue is the tangent direction to the feature, with an eigengap of $O(R/r)$.

3 Stability of the VCM

The analysis of the two previous paragraph shows that when S is a piecewise smooth hypersurface, the convolved VCM $\mathcal{V}_{S,R} * \chi_r$ carries information about the

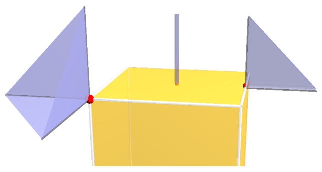


Figure 2.1: The shape of $\mathcal{N}_p S$ for various $p \in C$.

normals, principal curvature directions and sharp feature directions. The goal of this section is to study how replacing the surface S by a point cloud approximation C affects the VCM.

In order to do that, we will prove a continuity result for the map $K \mapsto \mathcal{V}_{K,R} * \chi$, from compact subsets of $\mathbb{R}^d$ to tensor-valued functions. The set of compact subsets of $\mathbb{R}^d$ is endowed with the *Hausdorff distance* $\mathrm{d_H}$: given two compact K and K', $\mathrm{d_H}(K, K')$ is by definition the smallest $\varepsilon > 0$ such that both $K \subseteq K'^{\varepsilon}$ and $K' \subseteq K^{\varepsilon}$. The operator norm of a symmetric matrix M is $\|M\|_{\mathrm{op}} = \sup_{V\in\mathbb{R}^d} \|V\|^{-2} V^{\mathbf{T}} M V$; the distance between two convolved VCM $\mathcal{V}_{K,K^R} * \chi$ and $\mathcal{V}_{K',K'^R} * \chi$ is measured by

$$\|\mathcal{V}_{K,K^R} * \chi - \mathcal{V}_{K',K'^R} * \|_\infty \\ := \sup_{p\in\mathbb{R}^d} \left\|\mathcal{V}_{K,K^R} * \chi(p) - \mathcal{V}_{K',K'^R} * \chi(p)\right\|_{\mathrm{op}}$$

The main theoretical result we use is the following Projection Stability Theorem from [3]:

Theorem 1 *Let K and K' be two compact sets of $\mathbb{R}^n$, and E an open set with rectifiable boundary. Then there is a constant $C(d, K, E)$ such that:*

$$\int_E \|p_K(x) - p_{K'}(x)\| \,\mathrm{d}x \leqslant C(d, K, E) \mathrm{d}_H(K, K')^{1/2}$$

As a consequence of this theorem, we will prove the Hausdorff-robustness of the convolved VCMs when the convolution kernel $\chi : \mathbb{R}^d \to \mathbb{R}$ is k-Lipschitz and bounded by 1 — this means that there exists a $k > 0$ such that $\forall(x, y),\ \|\chi(x) - \chi(y)\| \leqslant k\|x - y\|$ and $\forall x,\ |f(x)| \leqslant 1$.

Theorem 2 *If $\chi : \mathbb{R}^d \to \mathbb{R}$ is a bounded k-Lipschitz function, for every compact set $K \subseteq \mathbb{R}^d$ and $R > 0$, there is a constant $C'(d, K, R)$ such that for every other $K' \subseteq \mathbb{R}^d$,*

$$\|\mathcal{C}_{K,R} * \chi - \mathcal{C}_{K',R} * \chi\|_\infty \leqslant C'(d, K, R)\mathrm{d}_H(K, K')^{1/2}$$

Proof. We let E be the symmetric difference between K^R and K'^R. By Corollary II.5 in [3], we know that the volume $K^R \backslash E$ is in $O(\mathrm{d}_H(K, K'))$. Hence, letting $\delta(x) = x - \mathrm{p}_K(x)$, we can make the approximation,

$$\mathcal{V}_{K,R} * \chi(p) \simeq \int_E \delta(x)\delta(x)^{\mathbf{T}} \chi(\mathrm{p}_K(x) - p)\mathrm{d}x$$

The goal is now to bound the operator norm of the difference: $M = \int_E P(x) - P(x)\mathrm{d}x$, where $P(x) = \delta(x)\delta(x)^{\mathbf{T}}\chi(\mathrm{p}_K(x) - p)$ and P' is defined accordingly.

$$P(x) - P(x) \\ = \chi(\mathrm{p}_K(x) - p) \times \left(\delta(x)\delta(x)^{\mathbf{T}} - \delta'(x)\delta'(x)^{\mathbf{T}}\right) \\ + \left(\chi(\mathrm{p}_K(x) - p) - \chi(\mathrm{p}_{K'}(x) - p)\right)\delta'(x)\delta'(x)^{\mathbf{T}}$$

The operator norm of the first term of this expression is bounded by:

$$\|(2x - \mathrm{p}_K(x) - \mathrm{p}_{K'}(x))(\mathrm{p}_{K'}(x) - \mathrm{p}_K(x))^{\mathbf{T}}\|_{\mathrm{op}} \\ \leqslant 2R\,\|\mathrm{p}_{K'}(x) - \mathrm{p}_K(x)\|$$

By the k-Lipschitz property of χ, the norm of the second term can be bounded by

$$|\chi(\mathrm{p}_K(x) - p) - \chi(\mathrm{p}_{K'}(x) - p)|R^2 \\ \leqslant kR^2\,\|\mathrm{p}_K(x) - \mathrm{p}_{K'}(x)\|$$

Hence,

$$\|P(x) - P'(x)\|_{\mathrm{op}} \leqslant (2R + kR^2)\,\|\mathrm{p}_K(x) - \mathrm{p}_{K'}(x)\|$$

Integrating and using theorem 1 yields the bound

$$\|M\|_{\mathrm{op}} \leqslant C(d, K, E) \times [2R + kR^2]\mathrm{d}_H(K, K')^{1/2} \quad \square$$

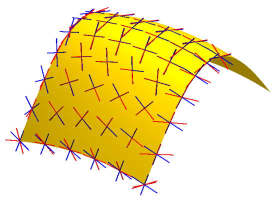
If S is a piecewise smooth surface, p a point of S and C_n a sequence of point clouds converging to S in the Hausdorff sense, the stability theorem says that $\mathcal{V}_{C_n,R} * \chi_r(p)$ converges to $\mathcal{V}_{S,R} * \chi_r(p)$ w.r.t the operator norm, and quantifies the speed of convergence.

Using the standard results of matrix perturbation theory (see e.g. [6]), one then obtains the convergence of the eigenvalues and eigenvectors of $\mathcal{V}_{C_n,R} * \chi_r(p)$ to those of $\mathcal{V}_{S,R} * \chi_r(p)$, the speed of convergence depending on the eigengap.

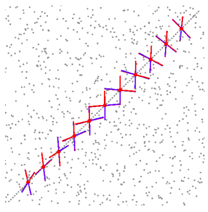
Since at a smooth point $p \in S$, the eigenvalues of $\mathcal{V}_{S,R} * \chi_r$ are 1, $\kappa_1(p)r/2$ and $\kappa_2(p)r/2$, one can expect a faster convergence rate for the estimated normal, and a faster convergence of the estimated principal curvature directions at points where the principal curvatures are very different.

4 Algorithm and experimental results

We propose two algorithms to compute the VCM of a point cloud in practice: a Monte Carlo algorithm, which is easy to implement and extends to high dimensions, and a deterministic algorithm based on tessellating the intersection between a Voronoi diagram and an offset. The second algorithm uses the fact that covariance matrices of tetrahedra have closed form expressions, which makes it fast in practice. Its implementation, however, is a little more intricate, and for this reason we only present the first algorithm.



(a) $z = \sin(3x) + \cos(x)$



(b) Sampling bias

Figure 4.2: (a) Parametric surface with exact (in blue) and computed (in red) principal curvature directions. (b) Principal directions under heavily biased sampling.

The Monte Carlo algorithm requires an additional parameter: N - the number of samples that are used. In practice, we observed good behavior with $N = 100n$, where n is the number of points in the point cloud. The convergence of the output of this algorithm to $\mathcal{V}_{C,C^R}$ with increasing N follows from the arguments presented in [3].

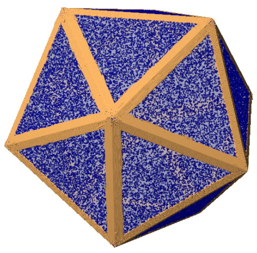
Algorithm 4.1 Monte-Carlo algorithm for VCM

```
  **Input:** a point cloud $C$, a scalar $R$, a number $N$
  **Output:** an approximation of $\mathcal{V}_{C,C^R} \simeq \frac{1}{N}\sum C_i\delta_{p_i}$
  **while** $k \leqslant N$ **do**
I.    Choose a random $p_i$ uniformly in $C$ ;
II.   Choose a random point $X$ uniformly distributed in $B(p_i, R)$ ;
II.   Find its closest point $p_j$ in $C$; then add the $d \times d$ matrix $\frac{1}{k}(X - p_j)(X - p_j)^{\mathbf{T}}$ to $C_i$, where $k$ is the number of points from $C$ in the ball $B(X, R)$.
  **end while**
```

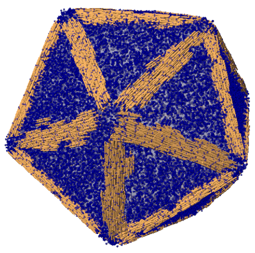
4.1 Experiments

We first tested our method on a parametric surface for which principal curvatures and principal curvature directions can be computed exactly. We sampled the function $z = \sin(3x) + \cos(y)$ with 100,000 points that were chosen uniformly at random within the domain $0 \leq x, y \leq 1$. Figure 4.2(a) shows this surface with the exact and computed principal curvature directions, using $R = 1$ and $r = 0.055$. Note that at the boundary, the computed directions follow the edges of the surface, making them useful for feature detection. To illustrate the performance of our algorithm under heavy sampling bias, we added 50,000 points in a small band along the diagonal in the domain. Figure 4.2(b) shows the results obtained with our method. The principal curvatures directions estimated by jet fitting [2] on this data set are strongly biased near the diagonal line; all methods based on least-square polynomial fitting will behave likewise.

To illustrate feature detection using our method, we randomly sampled 100,000 points on a unit icosahedron. We consider a point as a feature if the ratio



(a) no noise



(b) 10% noise

Figure 4.3: Features detected on a unit icosahedron with (a) no noise (b) 10% noise

of the second smallest to the smallest eigenvalue is greater than some threshold parameter. The results do not seem to be sensitive to this parameter, if it is within the range 10–40. Figure 4.3(a) shows the feature directions obtained by our method. Figure 4.3(a) shows the features detected after adding a vector chosen uniformly at random within a ball of radius 10% of the original radius to every point in the point cloud. The average angular deviation of the estimated feature directions to the real ones is 1.42° and 7.94° in the noiseless and noisy cases respectively.

5 Conclusion

In this paper, we have described a method for computing principal curvatures and principal curvature directions and detecting sharp features and feature directions on point cloud data with theoretical guarantees. We gave a bound on the deviation of the estimated directions from the real ones in terms of the Hausdorff distance between the point cloud and the underlying surface. We implemented and tested the method on multiple surfaces sampled with heavy bias and noise.

References

[1] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. Voronoi-based variational reconstruction of unoriented point sets. *Eurographics Symp. on Geom. Proc.*, 2007.

[2] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Eurographics Symp. on Geom. Proc.*, 2003.

[3] F. Chazal, D. Cohen-Steiner, and Q. Mérigot. Stability of boundary measures. *Arxiv preprint*, 2007.

[4] T. K. Dey and J. Sun. Normal and feature approximation from noisy point clouds. *FST&TCS*, 2006.

[5] H. Pottmann, J. Wallner, Y. Yang, Y. Lai, and S. Hu. Principal curvatures from the integral invariant viewpoint. *Comp. Aided Geom. Design*, 2007.

[6] G. Stewart and J. Sun. Matrix perturbation theory. *Computer Science and Scientific Computing*, 1990.

Inverted Ball as Boundary-constraint for Voronoi Diagrams of 3D Spheres

Martin Maňák* Ivana Kolingerová†

Abstract

We show how to avoid infinite cells in Voronoi diagrams of three-dimensional spheres. Infinite cells are unbounded cells at the convex-hull boundary of the input set. These cells can have Voronoi edges going to infinity or contain Voronoi vertices that are very distant from the convex hull. If our interest in parts of a diagram decreases as the distance from the convex hull increases or if we do not want to handle infinity, it might be useful to introduce a boundary constraint. We propose a constraint of extending the input set by a single *inverted ball*. When this ball encloses the entire input set, it stops any edge that would go to infinity otherwise. This approach only approximates the convex hull of the original input set. It is suitable for tracing-algorithms used for construction of these diagrams.

1 Introduction

Given a metric space and a finite input set of sites, Voronoi diagram partitions the space into cells. Each site gets its own cell. All points inside a cell are closer to the site owning the cell than to any other site. Intersecting cells constitute lower-dimensional elements of the diagram called Voronoi faces, edges and vertices.

Figure 1 shows an example of Voronoi diagram for a set of two-dimensional spheres – sites are circles here. Convex hull of the input set is shown as well. Cells that belong to the sites constituting the convex hull are partially unbounded, i.e., they extend themselves to infinity. However, there are also some cells that are bounded but their furthest boundary is quite distant from the convex hull with respect to its size.

If our interest in parts of a diagram decreases as the distance from the convex hull increases or if we do not want to handle infinity explicitly, it might be useful to introduce a boundary constraint.

A common technique is to add an extra site to the input set that represents infinity. Cells that were unbounded before become virtually bounded now. This introduces Voronoi elements at infinity and still allows elements at any distance from the convex hull.

*Faculty of Applied Sciences, University of West Bohemia, Pilsen, Czech Republic, manak@kiv.zcu.cz

†Faculty of Applied Sciences, University of West Bohemia, Pilsen, Czech Republic, kolingerova@kiv.zcu.cz

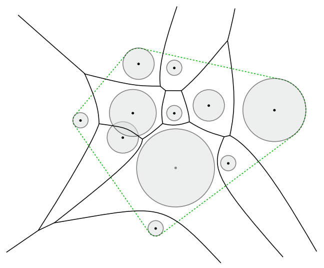


Figure 1: Voronoi diagram of spheres in 2D and its convex hull

This technique is used, e.g., in [1, 4, 6].

Another solution might be extending the input set by adding several extra sites. For example in 3D, four sites forming an outer tetrahedron are sufficient. These extra sites then bound cells that would otherwise extend themselves to infinity. They can also restrict some cells that are bounded but their boundaries are far away from the convex hull.

Another technique might be using the input set periodically or introducing some application-dependent constraint such as a cylinder as it is mentioned in [7].

In this paper, we introduce an *inverted ball* as a constraint for Voronoi diagrams of spheres which is nothing more than the complement of the interior of an ordinary sphere. It does not matter how many dimensions we are dealing with, only one extra site is sufficient for bounding all cells like in the case of the infinite site. Like the outer tetrahedron in 3D, the inverted ball plays a role of a finite constraint, i.e., coordinates of diagram elements are limited by a finite number. This might improve the robustness of some algorithms for diagram construction since we do not have to represent the whole domain of real numbers, just a small finitely bounded subset.

1.1 Euclidean Voronoi diagram for a set of spheres

This section briefly reviews the definition of Voronoi diagram for a set of spheres from [5]. These diagrams are often called additively weighted diagrams.

Let B be a finite set of spheres $B = \{b_1, \dots b_n\}$ in Euclidean metric space E^d of finite dimension d. We will call them as *balls*. Given an arbitrary point $x \in R^d$, we measure the signed distance of that point to a ball $b = (c, r)$ as $d(x, b) = ||x - c|| - r$ where $c \in R^d$ is the center and $r \in R$ is the radius of ball b. The sign of $d(x, b)$ is positive if the point is outside the ball, negative if the point is inside the ball and zero if it is on the boundary. The *Voronoi cell* for a ball b_i is the set of points

$$C_i = \{x \in R^d \,|\, \forall b_j \in B, j \neq i : d(x, b_i) \leq d(x, b_j)\}.$$

The *Voronoi diagram* for set B is the set of cells

$$VD(B) = \{C_1, \ldots, C_n\}.$$

We will denote Euclidean Voronoi diagrams for spheres as *VDS* and for points as *VDP*. In this paper, we will often omit the term Voronoi – for example, we will use only the term diagram instead of Voronoi diagram.

There are several algorithms for the construction of VDS. Edge-tracing [5], face-tracing [6] and the algorithm for three-dimensional Voronoi S-network [7] could be classified as tracing-algorithms. The basic idea of these algorithms is to treat the diagram as a graph of Voronoi vertices connected by edges. An algorithm finds some initial vertex at first. Four edge-trajectories are defined by this vertex. The algorithm then traces a trajectory in order to find the second vertex defining the edge. New vertices produce new trajectories and the tracing-step repeats.

There is also a region-expansion algorithm [4] which starts with a VDP constructed from centers of the input set of balls (it is also a valid VDS). Then it expands one ball after another, keeping the diagram to be a valid VDS by processing topological events.

2 Bounding diagrams by an inverted ball

Suppose we are given an input set of ordinary balls and want to create a constrained diagram from them. We can compute an approximation of their bounding sphere first, then magnify its radius and create an inverted ball from it. The inverted ball is just the complement of its interior. Then we insert the inverted ball to the input set and compute the diagram from this extended set.

A 2D example is shown in Figure 2. There is a diagram for a set of 2D balls with one inverted ball. Edges going to infinity were stopped by the inverted ball and some new edges were created. Note that unbounded cells as well as some long bounded cells were constrained by the inverted ball.

2.1 Distance function

We require that all balls from the original input set have non-negative radii. Then we can distinguish the inverted ball from them by a negative radius. This is not the same as negative weights in additively weighted diagrams! The distance function needs to be modified slightly for the inverted ball.

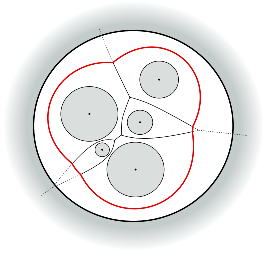


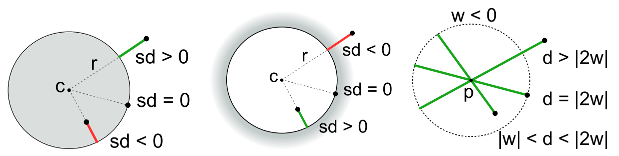
Figure 2: Inverted ball in 2D

Definition 1 *For ball $b = (c, r)$ and point x let the* signed distance *be defined as*

$$sd(b, x) = sgn_{0+}(r)(||c - x|| - |r|)$$

where $sgn_{0+}(r)$ is $+1$ for $r \geq 0$ and -1 for $r < 0$, $c \in R^3$ is the center and $r \in R$ is the signed radius of the ball.

Definition 1 extends the signed distance function from normal balls to inverted balls. It represents the shortest distance between a point and the boundary of a ball. Figures 3a and 3b show the signed distance for a normal ball and for an inverted ball, respectively. The function is negative for points inside the ball, positive outside and zero for points on the boundary. Figure 3c is the case of negative weight in additively weighted Voronoi diagrams. The meaning is different than the meaning of the signed distance.



(a) Normal ball (b) Inverted ball (c) Negative weight

Figure 3: Distance function, various configurations

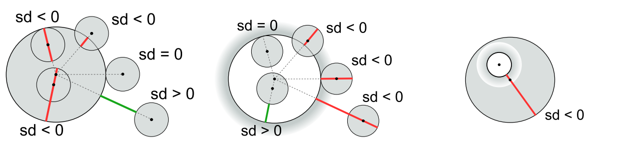
Now we can define the distance between two balls by reducing one of them to a point. In fact, we will select the ball with radius closer to zero, subtract the radius from the ball and add it to the other ball. Then we define the signed distance as in Definition 1. Reduction of an inverted ball implies sign change since all points (except the center) are considered inside of such a reduced inverted ball.

Definition 2 *For two balls $b_1 = (c_1, r_1)$ and $b_2 = (c_2, r_2)$ where at least one of them is not inverted, let the signed distance between them be defined as*

$$sd(b_1, b_2) = sgn_{0+}(r_i)sd(b, c_2)$$

where $b = (c_1, r_1 + r_2)$ is a modified ball, $i = 1$ if $|r_1| < |r_2|$ and $i = 2$ if $|r_1| \geq |r_2|$.

Several cases of signed distance between two balls according to Definition 2 are shown in Figure 4. The distance is measured between a big ball and a small ball. Figure 4a shows several configurations when only ordinary balls are involved. When a ball intersects another ball, the distance gets the negative sign and its magnitude represents the amount of penetration of one ball to another. When two balls just touch each other in their exterior, the distance is zero. When they do not intersect, the distance is positive. This is the same case as in ordinary VDS. Things get more complicated in Figure 4b since there is an inverted ball involved. This is almost the same configuration as in Figure 4a but the interior and exterior of the big ball got flipped and it became the inverted ball. The distance is different but the meaning is the same. In Figure 4c, there is an interesting configuration when the inverted ball has smaller magnitude than the ordinary ball. In this case, the distance is always negative since interiors of these balls always intersect.



(a) Ordinary balls (b) Inverted ball (c) Small inverted b.

Figure 4: Signed distance among balls

Definition 2 is useful for the definition of nearest neighbors among balls and for determining if one ball is fully contained in another ball.

2.2 The geometry of Voronoi faces and edges

It is known that the bisector between two ordinary balls is a hyperboloid of two sheets. In an ordinary VDS, a face is the maximal connected subset of such hyperboloid and edges are maximal connected subsets of conic sections. For more information to elements of VDS see [5]. But what if an inverted ball is involved?

Lemma 1 *Suppose we are given one inverted and one ordinary ball. If any of them is not fully contained in another ball, the set of points equidistant to these balls is an ellipsoid. At most one of its semi-axes has different length than the other.*

Proof. Denote the inverted ball as $b_1 = (c_1, r_1)$ and the ordinary ball as $b_2 = (c_2, r_2)$. Assume without loss of generality that $r_2 = 0$ (subtracting r_2 from radii of these balls does not change the set of equidistant points). Hence b_2 can be considered to be a point. Furthermore, it is sufficient to prove only the two-dimensional case in an arbitrary plane that contains both c_1 and c_2 as it is shown in Figure 5, i.e. that the set of equidistant points is an ellipse – denoted as f. This assumption is satisfied thanks to the rotation-symmetry with the axis defined by c_1 and c_2. The rotation-symmetry implies the shape of the ellipsoid, i.e. that at most one of its semi-axes has different length than the others – it is the rotation axis. It is easy to see that f is an ellipse: Consider an arbitrary point equidistant to both b_1 and b_2 with respect to the signed distance from Definition 1. It is the center of an empty sphere tangent to b_1 and b_2, denote it as $s = (c, r)$. Euclidean distance from c to c_1 equals to $r_1 - r$ and from c to c_2 equals to r. Their sum equals to r_1 which is constant for all points of f, hence f is an ellipse with foci c_1 and c_2. $\square$

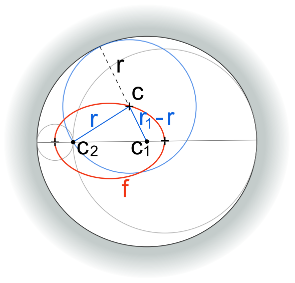


Figure 5: Elliptic bisector between an inverted ball and a ball with zero radius (in 2D)

Conjecture 1 *Suppose we are given one inverted and two ordinary balls. If any of them is not fully contained in another ball, the set of points equidistant to these balls is an ellipse.*

2.3 Vertex coordinates

For an ordinary d–dimensional VDS, coordinates of a vertex are the coordinates of the center of an empty sphere that is tangent to exactly $d+1$ balls. A sphere is empty when it does not intersect the interior of any ball. Their computation is described in [2, 3]. It leads to a system of d linear equations in $d+1$ variables and one quadratic equation in a free variable chosen from them. It turns out that almost the same approach can be used when an inverted ball is involved. We show this for $d = 3$.

Suppose we are given four balls $b_i = (c_i, r_i)$ for $i \in \{1 \dots 4\}$ where $c_i = (x_i, y_i, z_i) \in R^3$ is the center and $r_i \in R$ is the radius of ball b_i. These balls are allowed to intersect each other. We want to find a sphere that is tangent to all these balls and is empty. Let b_4 be the ball with the minimal positive radius and only b_1 may be the inverted ball (with $r_1 < 0$).

At first, we reduce all radii by r_4 and translate the system so that c_4 becomes the origin. Since r_4 is the minimal positive radius, b_4 reduces to a point, radii of b_1, b_2 and b_3 do not change signs. The transformation could be written as $b_i := b_i - b_4$. In the following text we suppose the system is already transformed, i.e. $b_4 = ((0,0,0),0)$. The problem is reduced to finding an empty sphere $s = (c, r)$ that touches the origin $(0,0,0)$ and is tangent to three balls b_1, b_2 and b_3. The sphere has a center $c = (x, y, z) \in R^3$ and a radius $r \in R$ – these are the unknowns.

Center c is the point equidistant to $b_1 \dots b_4$ in the sense of our signed distance sd from Definition 1. This can be written mathematically:

$$\forall i \in \{1 \dots 4\} : sd(c, b_i) = r$$

Expanding sd leads to an equivalent Eq. 1:

$$||c - c_i|| = sign_{0+}(r_i)(r + r_i) \tag{1}$$

The right-hand side of Eq. 1 will be non-negative. Powered by two, we get the same system as in [2]:

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = (r + r_1)^2 \tag{2}$$
$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = (r + r_2)^2 \tag{3}$$
$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = (r + r_3)^2 \tag{4}$$
$$x^2 + y^2 + z^2 = r^2 \tag{5}$$

The difference is that we can have $r_1 < 0$ for the inverted ball. Expanding Eq. 2, 3, 4 and subtracting Eq. 5 from them leads to the following system of three linear equations in four variables x, y, z and r.

$$\left(\begin{array}{cccc|c} x_1 & y_1 & z_1 & r_1 & (x_1^2 + y_1^2 + z_1^2 - r_1^2)/2 \\ x_2 & y_2 & z_2 & r_2 & (x_2^2 + y_2^2 + z_2^2 - r_2^2)/2 \\ x_3 & y_3 & z_3 & r_3 & (x_3^2 + y_3^2 + z_3^2 - r_3^2)/2 \end{array}\right) \tag{6}$$

When a free variable $f \in \{x, y, z, r\}$ is selected and the system is solved in terms of f, the result can be put into Eq. 5 in order to obtain a quadratic equation in terms of f. We can end up with two real solutions of s but are only interested in those which satisfy Eq. 1, i.e. the condition for non-negative right-hand side. If there were only ordinary balls with $r_i \geq 0$ then $r \geq 0$ would be satisfactory thanks to the choice of b_4. But we need also $sign_{0+}(r_i)(r + r_i) \geq 0$ since b_1 can be inverted.

The last step is to transform s back to the original system as $s := s + (c_4, -r_4)$ where c_4 and r_4 are from the original system. Note that this can lead to s with radius $r \in [-r_4, \infty)$ which can be negative. In this case it would be better to think about s as about a weighted point just like in additively weighted diagrams rather than it would be a sphere.

3 Results

Figure 6 shows results from the edge-tracing algorithm that has been modified to work with an inverted ball. First, there is the simplest case of four balls where one of them is inverted. The empty sphere of the initial vertex is shown as well. A more complex example is shown in the next figure (the inverted ball is hidden).

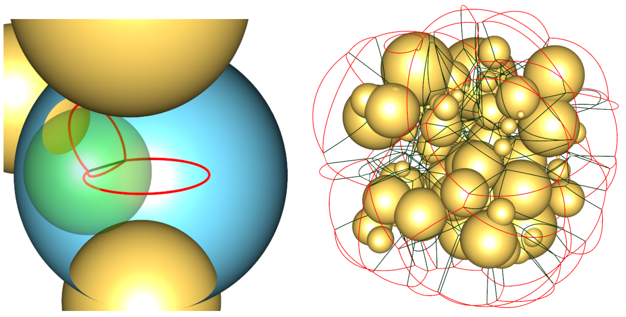


Figure 6: Inverted ball in three dimension

4 Conclusion

In this paper, we have shown how an inverted ball can be used as a boundary constraint for Voronoi diagrams of 3D spheres. We have proven that the inverted ball introduces ellipsoidal faces to this class of diagrams. This approach works well with edge-tracing and similar algorithms for construction of these diagrams but it might be problematic for other algorithms. We believe that it could simplify the tracing-based algorithms and make them more robust.

Acknowledgments

This work has been supported by the Grant Agency of the Czech Republic, project GA 201/09/0097.

References

[1] J.-D. Boissonnat and C. Delage. Convex hull and Voronoi diagram of additively weighted points. In G. S. Brodal and S. Leonardi, editors, *ESA*, volume 3669 of *Lecture Notes in Computer Science*, pages 367–378. Springer, 2005.

[2] M. L. Gavrilova. A reliable algorithm for computing the generalized Voronoi diagram for a set of spheres in the Euclidean d-dimensional space. In *CCCG*, pages 82–87, 2002.

[3] M. L. Gavrilova and J. Rokne. Updating the topology of the dynamic Voronoi diagram for spheres in euclidean d-dimensional space. *Comput. Aided Geom. Des.*, 20(4):231–242, 2003.

[4] D. Kim and D.-S. Kim. Region-expansion for the Voronoi diagram of 3D spheres. *Computer-Aided Design*, 38(5):417–430, 2006.

[5] D.-S. Kim, Y. Cho, and D. Kim. Euclidean Voronoi diagram of 3D balls and its computation via tracing edges. *Computer-Aided Design*, 37(13):1412–1424, 2005.

[6] D.-S. Kim, D. Kim, Y. Cho, and K. Sugihara. Quasi-triangulation and interworld data structure in three dimensions. *Computer-Aided Design*, 38(7):808–819, 2006.

[7] N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, and M. L. Gavrilova. An algorithm for three-dimensional Voronoi S-network. *Journal of Computational Chemistry*, 27(14):1676–1692, 2006.

Divide-and-Conquer for Voronoi Diagrams Revisited*

Oswin Aichholzer† Wolfgang Aigner† Franz Aurenhammer‡ Thomas Hackl† Bert Jüttler§
Elisabeth Pilgerstorfer§ Margot Rabl§

Abstract

We propose a simple and practical divide-and-conquer algorithm for constructing planar Voronoi diagrams. The novel aspect of the algorithm is its emphasis on the top-down phase, which makes it applicable to sites of general shape.

1 Introduction

The divide-and-conquer paradigm gave the first optimal solution for constructing the closest-site Voronoi diagram in the plane [13]. Though being a classical example for applying a powerful algorithmic method in computational geometry, the resulting algorithm became no favorite for implementation, not even in the case of point sites.

Literature tells us that divide-and-conquer is involved if emphasis is on the bottom-up phase, even if the sites are of relatively simple shape; see [10, 15, 5, 11]. The crux is the missing separability condition for the sites, which would prevent the merge curve from breaking into several components. Many alternative strategies for computing generalized Voronoi diagrams have been tried, including incremental insertion [3] and the plane-sweep technique [7].

In all these algorithms the bisector curves take part in the computation. Bisectors are usually composed of several curve pieces, and may even be two-dimensional if not defined carefully in the case of shared endpoints (which arise naturally when decomposing complex sites into simpler ones). Consequently, the algorithms are involved and also suffer from numerical imprecison. Difficulties may be partially eluded when working in the dual environment: Instead of intersecting two bisectors, the center of a circle tangent to the three defining sites is calculated. This bears the advantage of working on the sites directly. For general sites, however, tangent circles may not be unique, and are usually difficult to calculate.

The algorithm we propose works directly on the sites, too, but its atomic operation is much simpler, namely, an inclusion test of a site in a *fixed* circle. We first extract the combinatorial structure of the Voronoi diagram, and fill in the bisector curves later on. In contrast to existing Voronoi/Delaunay algorithms, no constructed object is ever discarded. Our setting is very general, including polygonal sites, circular disks, and even spline curves. Boundaries of curved planar objects with holes can be modeled.

Applications are manifold. One is motion planning in piecewise-circular environments [16] which, compared to piecewise-linear environments, leads to shorter and 'smoother' robot paths. Another is shape offsetting where, compared to similar approaches [9, 8, 4], our method is simpler because we compute only a combinatorial representation of the diagram for this application.

2 Dividing the Voronoi diagram

Our sites are pairwise disjoint topological disks of dimension two, one, or zero in the Euclidean plane $\mathbb{R}^2$. That is, a site is either homeomorphic to a disk or to a line segment, or is simply a point. This includes polygons, circular disks, and open spline curves as sites. Here and throughout this note, let S denote the given set of sites. The distance of a point x to a site $s \in S$ is $d(x, s) = \min_{y \in s} \delta(x, y)$, where δ denotes the Euclidean distance function. As done e.g. in [3, 15], we define the Voronoi diagram, $V(S)$, of S via its *edge graph*, $\mathcal{G}_S$, which is the set of all points having more than one closest point on the union of all sites. An edge of $\mathcal{G}_S$ containing points equidistant from two or more different points on the same site s is called a *self-edge* for s. The *regions* of $V(S)$ are the maximal connected subsets of the complement of $\mathcal{G}_S$ in $\mathbb{R}^2$. They are topologically open sets.

Observation 1 *The regions of $V(S)$ bijectively correspond to the sites in S. Each site is contained in its region, and regions are simply connected.*

We thus can talk of the region *of a site*, s, which we will denote with $R(s)$ in the sequel. The differences to a bisector-based definition of the Voronoi diagram should be noticed. Self-edges are ignored in such a definition unless the sites are split into suitable pieces. Such pieces, however, share boundaries—a fact that,

*Supported by FWF NRN 'Industrial Geometry' S9205-N12

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

‡Institute for Theoretical Computer Science, Graz University of Technology, Austria, auren@igi.tugraz.at

§Institute of Applied Geometry, Johannes Kepler University Linz, Austria, {Bert.Juettler,Margot.Rabl, Elisabeth.Pilgerstorfer}@jku.at

if not treated with care, may give rise to unpleasant phenomena like two-dimensional bisectors.

To get rid of the unbounded components of the diagram, we include a surrounding circle, Γ, (or any other desired curve) into the set S of sites. We can always choose Γ in a way such that each vertex of $V(S \setminus \{\Gamma\})$ is also a vertex of $V(S)$.

For later purposes, we seek a (minimal) set of points whose removal from the edge graph $\mathcal{G}_S$ breaks all its cycles. Finding such a set is nontrivial, in view of the possible presence of self-edges. For a site $s \neq \Gamma$, let $p(s)$ be a point on s with smallest ordinate, and denote with $q(s)$ the closest point on $\mathcal{G}_S$ vertically below $p(s)$. By the boundedness of $R(s)$, the point $q(s)$ always exists, and we define a new geometric graph as

$$\mathcal{T}_S = \mathcal{G}_S \setminus \{q(s) \mid s \in S \setminus \{\Gamma\}\}. \qquad (1)$$

Lemma 1 *The graph $\mathcal{T}_S$ is a tree.*

Proof. Omitted in this version. □

3 Augmented domains

Our next aim is to interpret the tree $\mathcal{T}_S$ in Lemma 1 as the medial axis of a generalized planar domain. In this way, we will be able to construct the Voronoi diagram $V(S)$ by means of a medial axis algorithm, as if a *simply connected* domain was the input. Usually, the similarity between these two structures is exploited the other way round: Medial axes are constructed as special cases of Voronoi diagrams.

Define $\mathcal{B} = B_0 \setminus \{s \in S \mid s \neq \Gamma\}$, where B_0 denotes the disk bounded by Γ. Then the medial axis, $\mathrm{MA}(\mathcal{B})$, of $\mathcal{B}$ is just the closure of the edge graph $\mathcal{G}_S$ of $V(S)$. We want to combinatorially disconnect the shape $\mathcal{B}$ at appropriate positions, such that the medial axis of the resulting domain corresponds to the tree decomposition $\mathcal{T}_S$ of $V(S)$.

As observed in [6], a maximal inscribed disk can be used to split the medial axis of a simply connected shape into two components which share a point at the disk's center. In order to extend this result to general shapes, we introduce the notion of an *augmented domain*, which is a set $\mathcal{A}$ together with a projection $\pi_{\mathcal{A}} : \mathcal{A} \to \mathbb{R}^2$. Initially, $\mathcal{A}$ is the original shape $\mathcal{B}$, and the associated projection $\pi_{\mathcal{B}}$ is the identity. Now, consider a maximal inscribed disk D of an augmented domain $\mathcal{A}$, which touches the boundary $\partial\mathcal{A}$ in exactly two points u and v. Denote with $\widehat{uv}$ and $\widehat{vu}$ the two circular arcs which the boundary of D is split into. The new augmented shape, $\mathcal{A}'$, which is obtained from $\mathcal{A}$ by splitting it with D, is defined as

$$\mathcal{A}' = \mathcal{A}^0 \cup D^1 \cup D^2$$

where $\mathcal{A}^0 = \{(x, 0) \mid x \in A \setminus D\}$, $D^1 = \{(x, 1) \mid x \in D\}$, and $D^2 = \{(x, 2) \mid x \in D\}$. The associated projection is

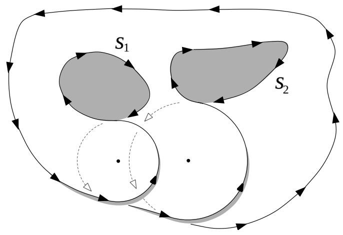


Figure 1: Boundary of an augmented domain.

$$\pi_{\mathcal{A}'} : \mathcal{A}' \to \mathbb{R}^2, \ (x, i) \mapsto \pi_{\mathcal{A}}(x).$$

We say that the line segment in $\mathcal{A}$ between points (x, i) and (y, j) is *contained* in $\mathcal{A}'$ if one of the following conditions is satisfied:

1. $i = j$ and the line segment $\overline{xy}$ avoids ∂D,
2. $\{i, j\} = \{0, 1\}$ and $\overline{xy}$ intersects the arc $\widehat{uv}$, or
3. $\{i, j\} = \{0, 2\}$ and $\overline{xy}$ intersects the arc $\widehat{vu}$.

For any two points (x, i) and (y, j) in $\mathcal{A}'$, their distance now can be defined. It equals the distance of $\pi_{\mathcal{A}}(x)$ and $\pi_{\mathcal{A}}(y)$ in $\mathbb{R}^2$, provided the connecting line segment is contained in $\mathcal{A}'$, and is ∞, otherwise. An (open) disk in $\mathcal{A}'$ with center (m, i) and radius ϱ is the set of all points in $\mathcal{A}'$ whose distance to (m, i) is less than ϱ. Such a disk is said to be *inscribed* in $\mathcal{A}'$ if its projection into $\mathbb{R}^2$ is again an open disk.

Having specified inscribed disks for $\mathcal{A}'$, the boundary of $\mathcal{A}'$ and the medial axis (transform) of $\mathcal{A}'$ can be defined as in the case of planar shapes. In particular, $\partial\mathcal{A}'$ derives from $\partial\mathcal{A}$ by disconnecting the latter boundary at the contact points u and v of the splitting disk D, and reconnecting it with the circular arcs $\widehat{uv}$ and $\widehat{vu}$. This process is depicted in Figure 1.

Concerning the medial axis, every maximal inscribed disk in $\mathcal{A}$ different from D corresponds to exactly one maximal inscribed disk in $\mathcal{A}'$, hence there is a bijection between $\mathrm{MAT}(\mathcal{A}) \setminus \{D\}$ and $\mathrm{MAT}(\mathcal{A}') \setminus \{D^1, D^2\}$. The medial axis of $\mathcal{A}'$ therefore is the same geometric graph as $\mathrm{MA}(\mathcal{A})$, except that the edge of $\mathrm{MA}(\mathcal{A})$ containing the center of D is split into two disconnected edges in $\mathrm{MA}(\mathcal{A}')$ which both have the center of D as one of their endpoints.

To draw the connection to the edge graph $\mathcal{G}_S$ of $V(S)$, the initial shape $\mathcal{B}$ is augmented with $|S| - 1$ maximal inscribed disks, namely, the ones centered at the points $q(s) \in \mathcal{G}_S$, where $q(s)$ was the vertical projection onto $\mathcal{G}_S$ of a point with smallest ordinate on the site s. Denote with $\mathcal{A}_S$ the resulting domain after these $|S| - 1$ augmentation steps. We may conclude the main finding of this section as follows.

Lemma 2 *The tree $\mathcal{T}_S$ in (1) is the medial axis of the augmented domain $\mathcal{A}_S$.*

4 The algorithm

Using Lemma 2, the Voronoi diagram $V(S)$ can be obtained by computing the medial axis of the augmented domain $\mathcal{A}_S$.

The construction of $\partial\mathcal{A}_S$ is easy once the augmenting disks are available. Their centers lie on the edge graph $\mathcal{G}_S$ of $V(S)$ but, of course, the disks need to be found without knowledge of $\mathcal{G}_S$. A simple and efficient plane-sweep can be applied; we omit the details here. The construction can be implemented in $O(n \log n)$ time if the sites in S are described by a total of n objects, each being managable in constant time. Note that $\partial\mathcal{A}_S$ then consist of $\Theta(n)$ pieces.

Given the fact that $\partial\mathcal{A}_S$ is highly self-crossing, it may seem complicated to compute the medial axis of $\mathcal{A}_S$. However, $\mathcal{A}_S$ has a connected boundary, and therefore can be split into subdomains with the same property using maximal inscribed disks. This suggests a divide-and-conquer algorithm for computing MA($\mathcal{A}_S$). The domain and its medial axis tree are split recursively, until directly solvable base cases remain. For simply connected shapes, a similar approach has been applied in [2, 1].

Recall that $\partial\mathcal{A}_S$ consists of pieces that bound inscribed disks (called *artificial arcs*) and pieces that stem from site boundaries (called *site segments*). To calculate a splitting disk, we fix some point p on a site segment and compute a maximal inscribed disk D for $\mathcal{A}_S$ that touches $\partial\mathcal{A}_S$ at p. Starting with an (appropriately oriented) disk of large radius, $\partial\mathcal{A}_S$ is scanned and the disk is shrunk accordingly whenever an intersection with a site segment occurs. Intersections with artificial arcs are, however, ignored. This works correctly because the set of maximal inscribed disks is the same for $\mathcal{A}_S$ and for the original shape $\mathcal{B}$ (except for finitely many augmenting disks). Computing a splitting disk takes $O(n)$ time, if each object describing the sites can be handled in $O(1)$ time.

5 Practical aspects

In view of keeping the algorithm efficient, disks that split the domain $\mathcal{A}_S$ in a balanced way are desired. Unfortunately, computing such a disk with simple means turns out to be hard. We can, however, choose a disk D randomly, by taking a random site segment on $\partial\mathcal{A}_S$ as its basis. Objects on $\partial\mathcal{A}_S$ and edges of MA($\mathcal{A}_S$) correspond to each other in an (almost) bijective way, which suffices to convey randomness from boundary objects to medial axis edges. For the analysis, we thus may suppose that the center c of D lies on every edge of MA($\mathcal{A}_S$) with the same probability. Under the assumption that the graph diameter of MA($\mathcal{A}_S$) is linear in n, the point c lies on the diameter with constant probability, and MA($\mathcal{A}_S$) is split at c into two parts of expected size $\Theta(n)$. A randomized runtime of $O(n \log n)$ results.

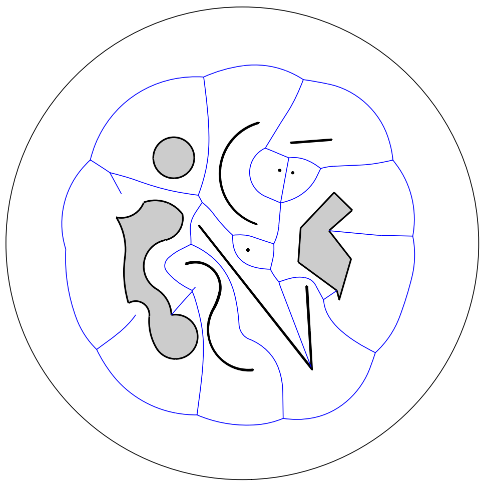


Figure 2: A mixed set of sites

n	atomic steps	ratio $n \log_2 n$	ratio $n(\log_2 n)^2$
507	6620	1.45	0.16
2070	32892	1.44	0.13
5196	91649	1.43	0.12
10474	199001	1.42	0.11
20488	417839	1.42	0.10
172198	4223178	1.41	0.09

Table 1: Count for complex sites bounded by n arcs

The assumption above is realistic in scenarios where a small number of sites is represented by a large number of individual objects. For example, if biarcs [12, 14] are used for approximation, then the number of leaves (hence also the number of vertices) of MA($\mathcal{A}_S$) is determined by the original sites and not by the number of biarcs used. See Table 1, where step counts for our algorithm are averaged (and rounded) over 40 different equal-sized inputs.

The other extreme is the case of n point sites. Here, by the way how $\mathcal{A}_S$ is constructed, the diameter of MA($\mathcal{A}_S$) will be typically much smaller, because many long 'vertical' branches will emanate from the surrounding circle Γ. As a simple heuristic, we may choose a small number of splitting disks tangent to Γ first, and continue with randomly splitting the result-

n	atomic steps	ratio $n \log_2 n$	ratio $n(\log_2 n)^2$
400	7591	2.20	0.25
2000	54662	2.49	0.23
4000	143391	3.00	0.25
20000	1015149	3.55	0.25
40000	2659149	4.35	0.28
200000	19820012	5.63	0.32

Table 2: Count for uniformly distributed point sites

ing augmented subdomains. See Table 2 for a sketch of the obtained results. We implemented the algorithm to accept circular arc input in its current version, including (though not optimizing) the handling of line segments and points. The Voronoi diagram in Figure 2 has been produced by this code.

The atomic step needed in the algorithm is an intersection test of a site-describing object and a given disk. This is among the simplest imaginable tests when a closest-site Voronoi diagram is to be computed by means of distance calculations. The structure and variety of the base cases depend on the type of sites. For point sites, there are only two of them, if the surrounding circle Γ is handled symbolically. They are of the simple form shown in Figure 3. (Artificial arcs are drawn dashed.) For circular arc splines, we get four generic base cases for C^1 continuity and nine additional cases for C^0 continuity; see [1].

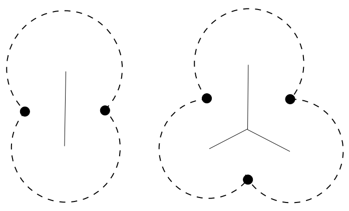


Figure 3: The two base cases for point sites.

6 Applications

We put particular emphasis on circular arcs as sites, because no practical algorithm for constructing their Voronoi diagram is available, and our algorithm naturally offers the ability to handle them. Moreover, biarcs [12, 14] enable a data-inexpensive and Voronoi diagram preserving approximation of general polynomial spline curves; we omit the details here.

The Voronoi diagram $V(S)$ of a set S of circularly approximated sites can be used as a tool for planning a robot motion in a piecewise-circular (PC)-environment [16]. Compared to piecewise-linear (PL)-environments, this offers several advantages. Not only can an approximation of $V(S)$ be computed more quickly now, but it also will consist of significantly fewer edges, namely, $\Theta(n^{\frac{2}{3}})$ instead of n. This leads to a more compact description of the paths the robot is supposed to move on. Another feature not shared by PL-environments is that the paths are locally C^1 between any two sites with C^1 boundaries, except for junctions with self-edges.

Several algorithms for shape offsetting are based on the Voronoi diagram or the medial axis [9, 8, 4]. Once more, a PC-representation of the input shape is advantageous, because the class of such shapes is closed under offsetting operations. Our Voronoi diagram algorithm is particularly well suited to the offsetting task, because it delivers the combinatorial structure without computing the edge graph explicitly.

References

[1] O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, and M. Rabl. Medial axis computation for planar free-form shapes. *Computer-Aided Design.* To appear.

[2] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Jüttler, M.Oberneder, and Z. Sír. Computational and structural advantages of circular boundary representation. In *Springer Lecture Notes in Computer Science*, volume 4619, pages 374–385, 2007.

[3] H. Alt, O. Cheong, and A. Vigneron. The Voronoi diagram of curved objects. *Discrete & Computational Geometry*, 34:439–453, 2005.

[4] L. Cao and J. Liu. Computation of medial axis and offset curves of curved boundaries in planar domain. *Comput. Aided Des.*, 40(4):465–475, 2008.

[5] L. Chew and R. Drysdale. Voronoi diagrams based on convex distance functions. In *Proc. 1st Ann. ACM Symposium on Computational Geometry*, pages 235–244, 1985.

[6] H. Choi, S. Choi, and H. Moon. Mathematical theory of medial axis transform. *Pacific Journal of Mathematics*, 181:57–88, 1997.

[7] S. Fortune. A sweep line algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[8] M. Held. Voronoi diagrams and offset curves of curvilinear polygons. *Computer-Aided Design*, 30(4):287–300, 1998.

[9] M. Held, G. Lukacs, and L. Andor. Pocket machining based on contour-parallel tool paths generated by means of proximity maps. *Computer-Aided Design*, pages 189–203, 1994.

[10] D. Kirkpatrick. Efficient computation of continuous skeletons. In *Proc. 20th Ann. Symp. Foundations of Computer Science*, pages 18–27, 1979.

[11] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Computational Geometry: Theory and Applications*, 3:157–184, 1993.

[12] D. S. Meek and D. J. Walton. Spiral arc spline approximation to a planar spiral. *J. Comput. Appl. Math.*, 107:21–30, 1999.

[13] M. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Ann. Symp. Foundations of Computer Science*, pages 151–162, 1975.

[14] Z. Šír, R. Feichtinger, and B. Jüttler. Approximating curves and their offsets using biarcs and Pythagorean hodograph quintics. *Comp.–Aided Design*, 38:608–618, 2006.

[15] C.-K. Yap. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, 2:365–393, 1987.

[16] C.-K. Yap and H. Alt. Motion planning in the CL-environment. In *Lecture Notes In Computer Science 382*, pages 373–380, 1989.

The Voronoi diagram of three arbitrary lines in $\mathbb{R}^3$

Hazel Everett[†] Christian Gillot[†] Daniel Lazard[‡] Sylvain Lazard[†] Marc Pouget[†]

Abstract

In this paper we study the Voronoi diagram of lines in $\mathbb{R}^3$. The Voronoi diagram of three lines in general position was studied in [8]. In this paper we complete this work by presenting a complete characterization of the Voronoi diagram of three arbitrary lines in $\mathbb{R}^3$. As in the general case, we prove that the arcs of trisectors are always monotonic in some direction and we show how to separate the connected components and to sort points along each arc of a trisector using only rational linear semi-algebraic tests. These results are important for the robust computation of the Voronoi diagram of polyhedra.

1 Introduction

Voronoi diagrams are ubiquitous objects of computational geometry. The most familiar and intuitive example is the Voronoi diagram of points in the plane with the Euclidean metric. Many extensions have been studied with higher dimension spaces, different metrics and/or other sites than points. For point sites in any dimension, the complexity of Voronoi diagrams is well understood. Optimal algorithms for their construction are known and robust efficient implementations are available (see for instance [1]). Still some important problems remain and are addressed in recent papers. For linear objects (segments and polygons) in two dimensions, the theory and implementations are also well studied [10].

In contrast, for lines, segments, and polyhedra in three dimensions much less is known. Even the combinatorial complexity of the Voronoi diagram of n lines or line segments in $\mathbb{R}^3$ is not yet settled. The best known lower (resp. upper) bound is $\Omega(n^2)$ (resp. $O(n^{3+\epsilon})$) [15]. From an algorithmic point of view, most of the previous work focused on computing the medial axis of a polyhedron, *i.e.,* a subset of the Voronoi diagram of the faces of the polyhedron [4, 12]. There have been many papers reporting algorithms for computing approximations of the Voronoi diagram (see for instance [5, 7]). On the other hand, as far as we know, only one algorithm has been proposed, following the paradigm of exact geometric computing [4]. This algorithm is nevertheless incomplete since it does not handle the case of singular one-dimensional Voronoi cells. Recently, some progress has been made on the related problem of computing arrangements of quadrics (each cell of the Voronoi diagram is a cell of such an arrangement) [2, 13, 14].

In this paper, we study the Voronoi diagram of three lines in $\mathbb{R}^3$ with the Euclidean metric. This is the first mandatory step on the way to design a robust and effective implementation of Voronoi diagrams of three-dimensional linear objects. We provide here a complete characterization of the Voronoi diagram of three arbitrary lines in $\mathbb{R}^3$. This completes the study of three lines, the case of three lines in general position having been studied in [8]. In particular, we identify those configurations having a singular one-dimensional Voronoi cell, information required to overcome the shortcoming of Culver's algorithm for the exact computation of the medial axis of a polyhedron [4].

Definitions and notations. The Voronoi diagram of a set of lines S is a partition of $\mathbb{R}^3$ into regions called cells associated with subsets s of S. The cell $V(s)$ consists of the set of points equidistant to the lines of s and strictly closer to those lines than to any other line. The distance between a point p and a line s, is defined as the minimum Euclidean distance between the point and any point on the line.

The Voronoi diagram of pairwise skew lines consists of cells of dimension k determined by $4-k$ lines, $0 \leqslant k \leqslant 3$. When the lines are no longer skew, a cell determined by $4-k$ lines may have dimension strictly smaller than k. For convenience, in the following a cell of dimension k (or kD cell) denotes a cell determined by $4-k$ lines, even if its dimension is smaller than k. In addition, we call bisec-

[†]LORIA - INRIA Lorraine - University Nancy 2, `Firstname.Name@loria.fr`

[‡]LIP6 - INRIA Rocquencourt - University Pierre et Marie Curie, `Firstname.Name@lip6.fr`

tor (resp. trisector) the dimension two (resp. one) cell of the Voronoi diagram of two (resp. three) lines. The cells are not necessarily connected even in the generic case [8] and when the lines intersect, the number of connected components, as we shall see, may increase. One way of handling intersecting objects is to subdivide them into non-intersecting pieces; for example, in the case of line segments in 2D, intersecting segments are partitioned into points and open segments [9]. Following this approach, one could handle intersecting lines by subdividing them into points, open rays and open segments. Since the bisector of two open segments is typically computed by first computing the bisector of their supporting lines, this approach also requires understanding the nature of the cells in the case of intersecting lines.

2 Results

Our main results are summarized in the following theorems which describe the cells of dimension one and two of the Voronoi diagram of three lines, see Table 1 for a more precise description.

Theorem 1 *The trisector of three lines is*

(i) a non-singular quartic if the three lines are pairwise skew but not all parallel to a common plane nor on the surface of a hyperboloid of revolution, (see fig. 1)

(ii) a cubic and a line if the three lines are pairwise skew and lie on the surface of a hyperboloid of revolution,

(iii) a nodal quartic if the three lines are pairwise skew and are all parallel to a common plane,

(iv) one or two parabolas or hyperbolas if there is exactly one pair of coplanar lines,

(v) between 0 and 4 lines if there are two pairs of coplanar lines.

In all cases, each branch of the trisector is monotonic in some direction.

Theorem 2 *A cell of dimension two of a Voronoi diagram of three lines consists of up to four connected components of hyperbolic paraboloids or planes.*

This characterization yields some fundamental properties of the Voronoi diagram of three lines which are likely to be critical for the analysis of the complexity and the development of efficient algorithms for computing Voronoi diagrams and medial axis of lines or polyhedra. In particular, we obtain the following:

Theorem 3 *There are rational linear semi-algebraic tests for*

(i) given a point on a 2D cell, deciding on which of the connected components of the cell it lies,

(ii) given a point on the trisector, deciding on which of the branches of the trisector it lies,

(iii) ordering points on each branch of the trisector, knowing that they lie on the trisector.

While the proofs primarily use basic notions from geometry and algebra, such as the classification of conics, it also relies on two recent results, the classification of quadric intersections [6], and a characterization of the common tangents to spheres [3].

3 Concluding remarks

We've given here a complete characterization of the Voronoi diagram of three lines. The results are necessary for the robust, exact computation of the Voronoi diagram of lines and the medial axis of polyhedra.

References

[1] F. Aurenhammer and R. Klein. Voronoi diagrams. In J. R. Sack and J. Urrutia, editors, *Handbook of computational geometry*, chapter 5, pages 201–290. Elsevier Publishing House, December 1999.

[2] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *SoCG'05*, pages 99–115, 2005.

[3] C. Borcea, X. Goaoc, S. Lazard, and S. Petitjean. Common tangents to spheres in $\mathbb{R}^3$. *DCG*, 35(2):287–300, 2006.

[4] T. Culver. *Computing the Medial Axis of a Polyhedron Reliably and Efficiently.* PhD thesis, University of North Carolina at Chapel Hill, 2000.

[5] T. K. Dey and W. Zhao. Approximate medial axis as a voronoi subcomplex. In *SMA '02*, pages 356–366, New York, NY, USA, 2002. ACM Press.

[6] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean. Near-optimal parameterization of the intersection of quadrics: II. A classification of pencils. *Journal of Symbolic Computation*, 2007. In press (24 pages).

[7] M. Etzion and A. Rappoport. Computing Voronoi skeletons of a 3-d polyhedron

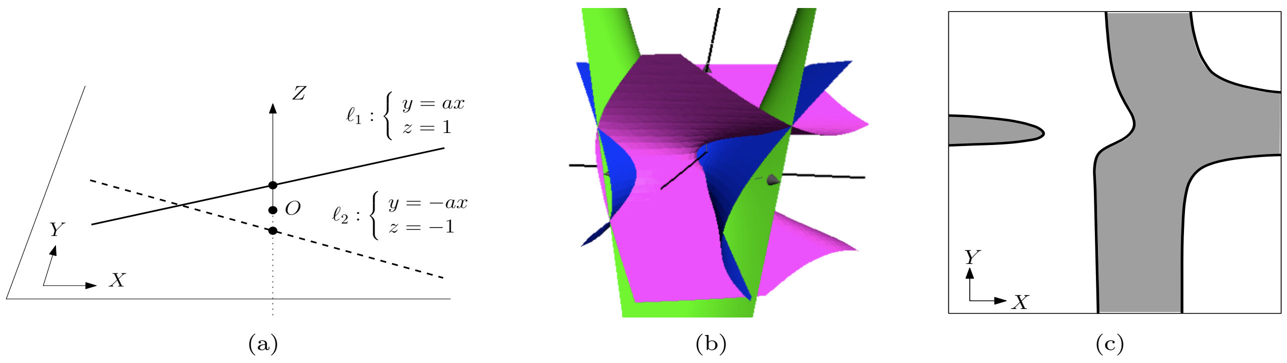
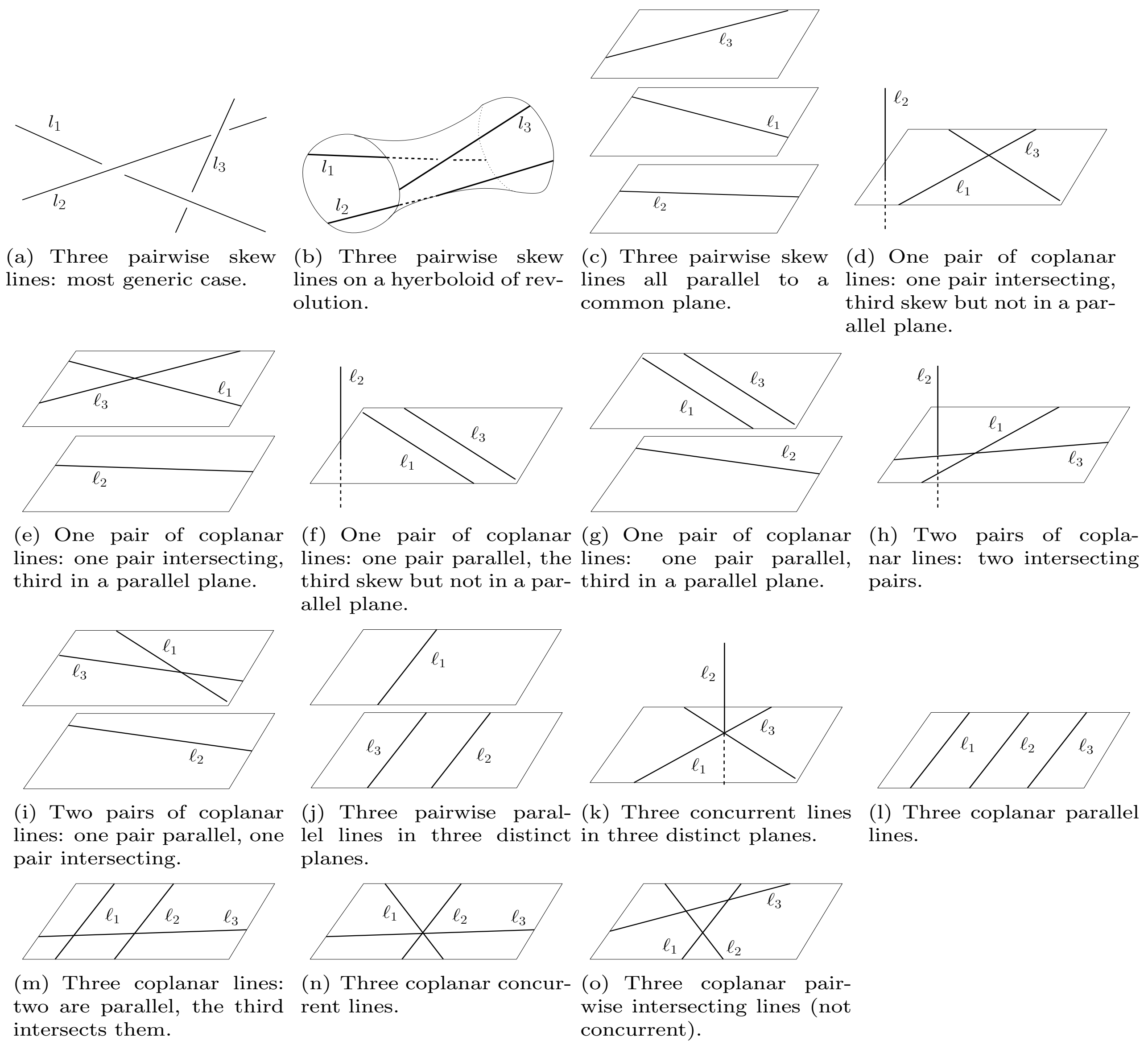

Z
$\ell_1 : \begin{cases} y = ax \\ z = 1 \end{cases}$
O
$\ell_2 : \begin{cases} y = -ax \\ z = -1 \end{cases}$
Y
X
(a)
(b)
Y
X
(c)

Figure 1: (a) The standard configuration of two skew lines. (b) The trisector of three skew lines, which is the intersection of two hyperbolic paraboloids, is a non-singular quartic. (c) Orthogonal projection of a 2D cell on a plane $\mathcal{P}$ with coordinate system (X, Y); the plane's normal is parallel to the common perpendicular of ℓ_1 and ℓ_2 and the X and Y-axes are parallel to the two bisector lines (in $\mathcal{P}$) of the projection of ℓ_1 and ℓ_2 on $\mathcal{P}$. The 2D cell is bounded by four branches of the trisector.

Configuration of lines	Trisector	Bisector type			No. CC		
		$\mathcal{H}_{1,2}$	$\mathcal{H}_{1,3}$	$\mathcal{H}_{2,3}$	$V_{1,2}$	$V_{1,3}$	$V_{2,3}$
Three pairwise skew lines							
Fig. 2(a) Not all parallel to a common plane nor on a hyperboloid of revolution	non-singular quartic	HP	HP	HP	2	2	2
Fig. 2(b) On a hyperboloid of revolution	cubic and a line	HP	HP	HP	2	2	2
Fig. 2(c) All parallel to a common plane	nodal quartic	HP	HP	HP	1	1	2
One pair of coplanar lines							
Fig. 2(d) One pair intersecting, third skew, not in a parallel plane	two hyperbolas^s	HP	HP	2P	3	3	2
Fig. 2(e) One pair intersecting, third in a parallel plane	two parabolas^s	HP	HP	2P	2	2	1
Fig. 2(f) One pair parallel, third skew, not in a parallel plane	one hyperbola	HP	HP	P	2(1)	1	1(2)*
Fig. 2(g) One pair parallel, third in a parallel plane	one parabola	HP	HP	P	1	1	1
Two pairs of coplanar lines							
Fig. 2(h) Two intersecting pairs	four lines^s	HP	2P	2P	4	3	3
Fig. 2(i) One pair parallel, one pair intersecting	two lines^s	HP	2P	P	2	1	2
Three pairs of coplanar lines in three distinct planes							
Fig. 2(j) Three pairwise parallel lines	one line	P	P	P	1	1	1
Fig. 2(k) Three concurrent lines	four lines^s	2P	2P	2P	4	4	4
Three coplanar lines							
Fig. 2(l) Three parallel lines	empty set	P	P	$\emptyset$	1	1	0
Fig. 2(m) Two parallel lines, the third one intersects them	two lines	P	2P	2P	2	1	1
Fig. 2(n) Three concurrent lines	one line	2P	2P	2P	2	2	2
Fig. 2(o) Three pairwise intersecting lines (not in a point)	four lines	2P	2P	2P	2	2	2

Table 1: Classification of the trisectors and 2D cells of three lines. Whenever there are two skew lines amongst the three lines, we assume they are labeled l_1 and l_2. In each case the bisectors are either hyperbolic paraboloids (HP), pairs of planes (2P), or single planes (P). The superscript symbol ^{s} indicates singular trisectors. *The number of connected components for cells $V_{1,2}$ and $V_{2,3}$ depends on whether the third skew line intersects the plane containing the parallel lines in between them or not.



(a) Three pairwise skew lines: most generic case.

(b) Three pairwise skew lines on a hyerboloid of revolution.

(c) Three pairwise skew lines all parallel to a common plane.

(d) One pair of coplanar lines: one pair intersecting, third skew but not in a parallel plane.

(e) One pair of coplanar lines: one pair intersecting, third in a parallel plane.

(f) One pair of coplanar lines: one pair parallel, the third skew but not in a parallel plane.

(g) One pair of coplanar lines: one pair parallel, third in a parallel plane.

(h) Two pairs of coplanar lines: two intersecting pairs.

(i) Two pairs of coplanar lines: one pair parallel, one pair intersecting.

(j) Three pairwise parallel lines in three distinct planes.

(k) Three concurrent lines in three distinct planes.

(l) Three coplanar parallel lines.

(m) Three coplanar lines: two are parallel, the third intersects them.

(n) Three coplanar concurrent lines.

(o) Three coplanar pairwise intersecting lines (not concurrent).

Figure 2: Classification of the degenerate cases.

by space subdivision. *CGTA*, 21(3):87–120, 2002.

[8] H. Everett, D. Lazard, S. Lazard, and M. S. E. Din. The Voronoi diagram of three lines in $\mathbb{R}^3$. In *SoCG'07*, pages 255–264, S. Korea, 2007.

[9] M. . Karavelas. A robust and efficient implementation for the segment voronoi diagram. In *Proc. Internat. Symp. on Voronoi diagrams in Science and Engineering*, pages 51–62, 2004.

[10] M. I. Karavelas.' A robust and efficient implementation for the segment Voronoi diagram. In *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 51–62, 2004.

[11] V. Koltun and M. Sharir. Three dimensional euclidean Voronoi diagrams of lines with a fixed number of orientations. *SIAM Journal on Computing*, 32(3):616–642, 2003.

[12] V. J. Milenkovic. Robust construction of the Voronoi diagram of a polyhedron. In *CCCG'93*, pages 473–478, 1993.

[13] B. Mourrain, J.-P. Técourt, and M. Teillaud. On the computation of an arrangement of quadrics in 3D. *CGTA*, 30(2):145–164, 2005.

[14] E. Schömer and N. Wolpert. An exact and efficient approach for computing a cell in an arrangement of quadrics. *CGTA*, 33(1-2):65–97, 2006.

[15] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *DCG*, 12:327–345, 1994.

Voronoi Diagram in the Klein model using Finsler Geometry

A. Laleh A. Mohades Z. Nilforoushan M. M. Rezaii*

Abstract

In this paper in order to compute the Voronoi diagram in Finslerian spaces, we consider a Finsler space with the Funk metric and exhibit the hyperbolic Klein model. Since in this space the geodesics are straight lines, our computations of Voronoi diagrams are done in the Klein model rather than Finslerian space.

1 Introduction

Given a set of sites and a distance function from a point to a site, a *Voronoi diagram* can be roughly described as the partition of the space into cells that are the locus of points closer to a given site than to any other site.

Voronoi diagrams belong to the computational geometry's favorite structures. They arise in nature and have applications in many fields of science [4]. Excellent surveys on the background, construction and applications of Voronoi diagrams can be found in Aurenhammer's survey [2] or the book by Okabe et al. [9]. In 1995, Onishi and Takayama proposed the construction of Voronoi diagram on the Upper half-plane, one of the 2-dimensional hyperbolic space models [10]. In addition, the second and third authors studied this idea on the other 2-dimensional hyperbolic model called Poincaré disk [8] and generalized 3-dimensional hyperbolic Voronoi diagrams [7]. In [10] the authors have mentioned to study the Voronoi diagram in *Finsler space* and this motivated us to do some research in this direction.

Finsler geometry in a sense could be interpreted as a generalization of the Riemannian geometry. In this geometry, the distance of two points will depend on two factors, location of points and direction. That is, the distance function is not necessarily symmetric. In this paper we will study a convex space with a certain Finslerian metric whose geodesics are straight lines (such metrics are very important to be studied, since they relate to Hilbert's fourth problem). This approach will lead to having the benefit of doing simpler computations to construct the Voronoi diagram in special Finslerian spaces.

*Faculty of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran. {aglaleh, mohades, nilforoushan, mmreza}@aut.ac.ir

2 The Finsler geometry

In this section we present some facts on the Finsler geometry.

Observation 1 *We use here the Einstein summation convention, i.e., $v^i\partial/\partial x^i$ represents $\sum_{i=1}^n v^i\partial/\partial x^i$ and so forth.*

Definition 1 *Let V be a finite dimensional vector space. A Minkowski function on V is a function $L : V \longrightarrow [0,\infty)$ which has the following properties:*
1. L is C^∞ on $V\backslash\{0\}$
2. L is positively homogeneous of degree two, i.e., $L(\lambda y) = \lambda^2 L(y), \quad \forall\lambda > 0, \quad \forall y \in V$.
3. For every $0 \neq y \in V$, the fundamental form g_y is non-degenerate, where
$g_y : V\times V \longrightarrow \mathbb{R}$,
$g_y(u,v) = \frac{1}{2}\frac{\partial^2}{\partial s\partial t}[L(y+su+tv)]\mid_{s=t=0}$.
A Minkowski space is such a pair (V,L).

Let M be a n-dimensional manifold with a local coordinates system $x = (x^1,...,x^n)$. Suppose TM is the tangent bundle of M with the natural projection $\pi : TM \longrightarrow M$. Let $(x,y) = (x^1,...,x^n,y^1,...,y^n)$ be a local coordinates system on TM. We denote a point $x \in M$ with its coordinates system (x^i) and a vector $y \in TM$ with its coordinates (x^i,y^i), if no confusion is occurred .

Definition 2 *Let M be an n-dimensional manifold. a Finsler metric on M is a function $L : TM \longrightarrow [0,\infty)$ which has the following two properties:*
1. L is C^∞ on $TM\backslash\{0\}$
2. For any $x \in M$, the restriction function $L_x := L\mid_{T_xM}$ is a Minkowski function on T_xM. A Finsler space is such a pair (M,L).

Let C be an oriented smooth curve in M given by the equations $x^i = x^i(t)$, $t \in [a,b]$, $i = 1,...,n$, with $A = (x^i(a))$ and $B = (x^i(b))$. Suppose AB is non-degenerate, that is, $L(x(t),\dot{x}(t)) \neq 0, \quad \forall t \in [a,b]$, where $x(t) = (x^i(t))$, and $\dot{x}(t) = (\frac{dx^i}{dt})$. The curve $C := (x^i(t))$ is called a geodesic of the Finsler manifold (M,L), if for any coordinate system (x,y) there exist a parameter s on C such that $x^i = x^i(s)$ is a solution of

$$\frac{d^2x^i}{ds^2} + \frac{\partial G^i}{\partial y^k}(x(s),x'(s))\frac{dx^k}{ds} = 0 \quad i = 1,2,.... \qquad (1)$$

Now let B^2 be the unit disc in $\mathbb{R}^2$ with the Euclidean norm $|.|$. For any two points $p, q \in B^2$, let $z_{pq} \in \partial B^2$ be the intersection point of the line passing through p and q with ∂B^2. Define $d(p,q) := \ln|\frac{z_{pq}-p}{z_{pq}-q}|$, then d is a non-reversible distance on B^2, i.e., $d(p,q) \leq d(p,r)+d(r,q)$, and $d(p,q) \neq d(q,p)$. Here d is called the Funk distance.

Definition 3 *Let d stand for the Funk distance on B^2. Define $\overline{d} : B^2 \times B^2 \longrightarrow [0,\infty)$ by $\overline{d}(p,q) = \frac{d(p,q)+d(q,p)}{2}$. $\overline{d}$ is a distance which is called the Klein distance.*

Note that we have $\overline{d}(p,q) = \overline{d}(q,p)$, but $d(p,q) \neq d(q,p)$.

Let d and $\overline{d}$ be the Funk and the Klein distances on B^2, if $X = (x,y) \in B^2$ and $Y = (u,v) \in TB^2 - \{0\} = \mathbb{R}^2 - \{0\}$, let $F_X(y) = \lim_{\varepsilon \longrightarrow 0^+} \frac{d(X,X+\varepsilon Y)}{\varepsilon}$ and $\overline{F}_X(y) = \lim_{\varepsilon \longrightarrow 0^+} \frac{\overline{d}(X,X+\varepsilon Y)}{\varepsilon}$. Then by direct calculation we obtain that

$$F = \frac{\sqrt{(xu+yv)^2+(u^2+v^2)(1-x^2-y^2)}+xu+yv}{1-x^2-y^2},$$
$$\overline{F} = \frac{\sqrt{(xu+yv)^2+(u^2+v^2)(1-x^2-y^2)}}{1-x^2-y^2},$$

moreover we have, $F_x = FF_u$, $F_y = FF_v$.

Let (S,F) be a Finsler surface. For a curve C on S issuing from p to q, the length of C is given by $\mathcal{L}(C) = \int_a^b F(\dot{\alpha}(t))dt$, where $\alpha : [a,b] \longrightarrow S$ is the coordinate map of C with $\alpha(a) = p$ and $\alpha(b) = q$.

Assume that C is covered by a coordinate map $\varphi : D \longrightarrow S$. We can express $\alpha(t)$ by $\alpha(t) = \varphi(x(t), y(t))$. Put $L(x,y,u,v) := \frac{1}{2}F^2(Y)$, $\quad Y = u\varphi_x + v\varphi_y$. By the methods in the calculus of variation we know that the functions $x(t)$ and $y(t)$ satisfy the following second order differential equations:

$$x''(t) + 2G(x(t), y(t), x'(t), y'(t)) = 0, \tag{2}$$

$$y''(t) + 2H(x(t), y(t), x'(t), y'(t)) = 0, \tag{3}$$

where $G = G(x,y,u,v)$ and $H = H(x,y,u,v)$ are given by

$$G := \frac{(L_xL_{vv} - L_yL_{uv}) - (L_{xv} - Lyu)L_v}{2(L_{uu}L_{vv} - L_{uv}L_{uv})},$$
$$H := \frac{(-L_xL_{uv} - L_yL_{uu}) + (L_{xv} - Lyu)L_u}{2(L_{uu}L_{vv} - L_{uv}L_{uv})}.$$

We call G and H the geodesic coefficients of F.

Lemma 1 *Let (S,F) be a Finsler surface and $\varphi : D \longrightarrow S$ a coordinate map such that the geodesic coefficients G and H are in the forms $G = P(x,y,u,v)u$, $\quad H = P(x,y,u,v)v$. Then any path on the surface must be the image of a straight line under φ.*

Proof. We provide a proof in the full paper. □

Definition 4 *Let (S,F) be a Finsler surface. The Gaussian curvature of S is defined by*

$$K(Y) = \frac{1}{F^2(Y)}\{2G_x + 2H_y + 2G_uH_v - 2H_uG_v - Q^2 - Q_xu - Q_yv + 2GQ_u + 2HQ_v\},$$

where $Q := G_u + H_v$.

Now by direct calculation and using Lemma 1, we have $K = -1$ (we provide a proof in the full paper). Thus geodesics of F are straight lines.

Hence by considering the Funk metric for the Finsler space, one can reach to the Klein model as one of the hyperbolic space models. The curvature of this model is -1 and its geodesics are straight lines. In the next section we study this model in detail.

3 Geometric structures in the Klein model

In this section we will investigate one of the hyperbolic geometry models which built within Euclidean geometry, that maintains just the Euclidean notion of lines but not angles. In this model, points are the set of points inside the unit circle B^2 and lines will be any chord of the boundary of B^2 minus the end points. The points which lie on the boundary of B^2 are treated as the points-at-infinity. In the following we listed some required notions of the Klein model.

3.1 Line

For any two points A and B inside B^2, the Euclidean chord passing through A and B is the line connecting A and B.

3.2 Distance

To compute the distance between two points A and B, construct the Euclidean chord connecting A and B and assume that it intersects ∂B^2 at the points P and Q. Using the Euclidean length of $\overline{AP}$, $\overline{AQ}$, $\overline{BP}$ and $\overline{BQ}$ we define the Klein distance between A and B by $d_K(A,B) = \frac{1}{2}\ln\frac{\overline{AP}.\overline{BQ}}{\overline{AQ}.\overline{BP}}$.

3.3 The pole of a Klein line

Let l be a line and assume that l intersects the boundary of B^2 at two points Q and R. Then the intersection of the tangent lines at Q and R to the boundary of B^2 which will be stood for $P(l)$, is a point and is called the pole of line l.

3.4 The construction of a Klein line perpendicular to the given Klein line AB through a given point C lying on AB

Using the pole of line AB, say P, draw the Euclidean line connecting C and P. PC intersects the boundary of B^2 at two points, say M and N. Then the chord MN will be perpendicular to the given line AB at the points of intersection.

3.5 Midpoint of a Klein segment

Let A, B be two points inside the B^2 and P be the pole of Klein line AB. The Euclidean lines PA and PB intersect the boundary of B^2 at four points, say S, S', T and T'. Therefore the intersection point of two chords ST' and $S'T$ is the midpoint of the segment AB.

3.6 Perpendicular bisector

By combining 7 and 9, we can construct the perpendicular bisector of the given segment.

3.7 Circle

There is an isomorphism between the Klein model and Poincaré disk[1] model that takes the basic objects of the Klein model in to the basic objects in the Poincaré disk, which preserves the basic relationships (e.g., incidence congruence, betweenness, etc.) and vice versa:

$$\phi : B^2 \longrightarrow D, \quad \phi(z) = \frac{2z}{1+|z|^2} \tag{4}$$

In order to construct the Klein circle centered at point O with radius OP (for a given point P in B^2), let $O' = \phi(O)$ and $P' = \phi(P)$ be the corresponding points in D. Then construct the Poincaré circle centered at O' with radius $O'P'$, say C'. Hence, our desired Klein circle is $\phi^{-1}(C')$.

Now if O is the center of a Klein circle and P is a point on it, we use the isomorphism ϕ and find the images $O' = \phi(O)$ and $P' = \phi(P)$, which O' and P' are in the Poincaré disk D. Next, construct the Poincaré circle centered at O' with radius $O'P'$, say C_1. Therefore, if X' is an arbitrary point on C_1, by using ϕ^{-1}, we have $X = \phi^{-1}(X')$ in the Klein model B^2. Hence the Klein circle C_2 centered at O with radius OP, is the locus of points X as the point X' moves around circle C_1.

For more details on the mentioned notions of the Klein model, see [5].

[1]The Poincaré disk model is another two-dimensional model for hyperbolic geometry which is defined as the disk $D = \{(x,y) \in \mathbb{R}^2 | x^2 + y^2 < 1\}$, with hyperbolic metric $ds^2 = \frac{dx^2+dy^2}{(1-x^2-y^2)^2}$; see [1] for more details.

Observation 2 *We notice that B^2 and D are the same sets, but we choose two different symbols to emphasize that we have two different models (with different metrics).*

4 Basics of Voronoi diagrams in the Klein model

Most of the material and terminology contained in this section has been presented in [6]. We include it here for the convenience of the reader and study in the Klein model.

The definition of the Voronoi diagram, that is sometimes called the metric fundamental polygon, is rather simple. Think of a metric space (B^2, d_K) and some sites $S = \{p_1, ..., p_n\} \subset M$.

Definition 5 *Given two sites $p, q \in B^2$, the bisector $B(p,q)$ consists of all points $x \in B^2$ that have the same Klein distance to p as to q, i.e.*

$$B(p,q) = \{x \in B^2 \mid d_K(p,x) = d_K(q,x)\}. \tag{5}$$

When $S = \{p_1, p_2\}$, the bisector gives the Voronoi diagrams of S. In general, any Voronoi edge is the part of a bisector. A bisector in B^2 has the following impressive property.

Lemma 2 *For any two points $p_1, p_2 \in B^2$, there exist a bisector for them in B^2 and it would be a geodesic of B^2*

Proof. Using Subsection 3.6 and the fact that any chord of the boundary of B^2 is a geodesic of B^2, the proof is immediate. □

In addition, the metric d_K makes it possible to define half-spaces as in the Euclidean case.

Definition 6 *Let $p, q \in B^2$. $D(p,q)$ is the set of all points that are closer to p than to q:*

$$D(p,q) = \{x \in B^2 \mid d_K(p,x) < d_K(q,x)\}. \tag{6}$$

An analogue characterization can be made for q:

$$D(q,p) = \{x \in B^2 \mid d_K(p,x) > d_K(q,x)\}. \tag{7}$$

Hence for any bisector $B(p,q)$, each of the two connected components $D(p,q)$ and $D(q,p)$ of $B^2 \backslash B(p,q)$ is called half-space.

Lemma 3 *Half-space is a convex set.*

Proof. We provide a proof in the full paper. □

Now it is possible to declare Voronoi regions and the Voronoi Diagram:

Definition 7 *$VR(p,S) = \bigcap_{q\in S\setminus\{p\}} D(p,q)$ is called Voronoi region of p with respect to S. Furthermore $VR(S) = \bigcup_{p\in S} \partial VR(p,S)$, is titled Voronoi diagram of the reference point set S.*

Lemma 4 *For any point $O(x,y) \in B^2$, the set of points with constant Klein distance from the point O is a circle.*

Proof. Using Subsection 3.7 and [8], the proof is immediate. □

Lemma 5 *For given three distinct points in B^2, the set of points which has the same Klein distance from three points, if exists, is a point.*

Proof. Using Subsection 3.7 and [8], the proof is immediate. □

Definition 8 *A set $C \subset B^2$ is called strongly star shaped if and only if there exists a center point $p \in C$ such that for every point $q \in C$ every shortest geodesic between p and q belongs to C.*

This yields the following

Theorem 6 *Consider the Klein model B^2 with its intrinsic distance function d_K. Every Voronoi region $VR(p,S)$ is strong star shaped with center p.*

Proof. We bring the proof it in the full paper. □

A direct consequence of this theorem is stated in the next result.

Corollary 7 *Let B^2 be the Klein model and d_K the Klein distance function. Every Voronoi region is connected.*

Theorem 8 *Let (B^2, d_K) be the Klein metric space and S the set of sites. Then every Voronoi region $VR(p_i,S)$ is open and $V(S)$ is closed.*

Proof. We provide a proof in the full paper. □

5 Construction of the Voronoi diagrams in the Klein model using plane sweep algorithm

In this paper we will use the plane sweep algorithm. In [3], Dehne and Klein gave the first deterministic algorithm for computing the Voroni diagram of n points in an arbitrary nice metric in the plane within $O(n \log n)$ time and linear space and among the efficient deterministic algorithms, they chose the plane sweep algorithm which might be the easiest one to implement.

Since the Klein model B^2 is a subset of the plane, it is sufficient to show that d_K enjoys all the conditions of nice metric.

Definition 9 *A metric d on $\mathbb{R}^2$ is called nice if:*
1. each d-circle contains a standard circle, and vice versa,
2. each d-circle is contained in a standard circle,
3. for any two points a and c there exists a point $b \notin \{a,c\}$ such that $d(a,c) = d(a,b) + d(b,c)$ holds,
4. if p, r are two points, or a point and a line, then the boundary of $B(p,r)$ consists of two curves each of which is homeomorphic to a line. The intersection of two such curves consists of finitely many connected components..

Lemma 9 *The Klein metric d_K is a nice metric.*

Proof. We provide a proof in the full paper. □

Now by Theorem 16 of [3] and Lemma 9, the following Corollary will be attained.

Corollary 10 *The Voronoi diagram of n points in B^2 can be computed by the plane sweep algorithm in optimal time $O(n \log n)$, using linear space.*

References

[1] J. W. Anderson. Hyperbolic Geometry. New York. Springer-Verlag, 1999.

[2] F. Aurenhammer and R. Klein. Voronoi Diagrams. In: J. R. Sack and G. Urrutia (Eds.), Handbook on Computational Geometry, Elsevier, pp. 201–290, 1999.

[3] F. Dehne and R. Klein. "The Big Sweep":On the Power of the Wavefront Approach to Voronoi Diagrams. Algorithmica, 17(1), pp. 19–32, 1997.

[4] S. Drysdale. Voronoi diagrams: Applications from Archaology to Zoology. Regional Geometry Institute, Smith College, July 19, 1993.

[5] M. Greenberg. Euclidean and Non-Euclidean Geometries: Development and History. W. H. Freeman and Company, Newyork, 3rd Edition, 1993.

[6] R. Klein. Concrete and abstract Voronoi diagrams. LNCS 400, Springer-Verlag, Berlin, 1989.

[7] Z. Nilforoushan, A. Laleh, M. M. Rezaii and A. Mohades. 3-D Hyperbolic Voronoi Diagrams. Submitted to Computer-Aided Design, 2006.

[8] Z. Nilforoushan and A. Mohades. Hyperbolic Voronoi Diagram. ICCSA 2006, LNCS 3984, pp. 735–742, 2006.

[9] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley Series in Probability and Statistics, 2000.

[10] K. Onishi and N. Takayama. Construction of Voronoi diagram on the Upper half-plane. In IEICE TRANS. Fundamentals. Vol. E79-A, 2, pp. 533–539, 1995.

A Practice-Minded Approach to Computing Motorcycle Graphs[*]

Stefan Huber[†] Martin Held[‡]

Abstract

We study the computation of motorcycle graphs and give the first formal definition of the motorcycle graph as a set of constraints rather than as the result of some process. A constructive proof that the constraints can be fulfilled is cast into a simple algorithm for computing motorcycle graphs, with geometric hashing used for speeding up the algorithm. Extensive practical tests of the C++ implementation of our algorithm on over 22 000 data sets provide the surprising practical evidence that it processes n motorcycles in $O(n \log n)$ time on average. This observation is backed by a stochastic analysis which reveals that our hash-based algorithm can be expected to run in $O(n\sqrt{n} \log n)$ time, provided that the motorcycles are sufficiently uniformly distributed in the plane.

1 Introduction

1.1 Motivation

Consider n motorcycles in the plane, each starting at some point and driving with a constant speed along a straight path. As a motorcycle proceeds it leaves a trace behind it on the plane. Every motorcycle crashes when it reaches the trace left behind by another motorcycle: it stops moving, but its own trace remains. Roughly speaking, the motorcycle graph is the union of the resulting motorcycle traces.

Motorcycle graphs were introduced by Eppstein and Erickson [3], who also presented an algorithm which runs in $O(n^{17/11+\epsilon})$ time. Motorcycle graphs are known to be related to straight skeletons: Cheng and Vigneron [1] studied a straight-skeleton algorithm for simple polygons that computes the motorcycle graph in a preprocessing step in $O(n\sqrt{n} \log n)$ worst-case time. Motorcycle graphs are also related to other problems like visibility and art gallery problems, see [2, 3]. Recently, Eppstein et al. [4] introduced motorcycle graphs on quad literal meshes for extracting canonical partitions of those. However, despite of their practical usefulness no implementation of a sub-quadratic algorithm is known.[1]

1.2 Our contribution

We suggest an algorithm for computing motorcycle graphs that is both easy to implement and fast. We start with a formal definition of the motorcycle graph which does not define the motorcycle graph as the outcome of some process, and argue that our definition conforms to the one given by Eppstein and Erickson. Subsequently, we show how to apply geometric hashing in order to get a simple and easy-to-implement algorithm for computing motorcycle graphs.

Even though the motorcycles might move across large portions of the hash grid during the course of computation, our C++ implementation exhibits a surprisingly good performance in over 22 000 practical tests, involving tests with more than a million motorcycles. More precisely, in the vast majority of our tests we achieve an $O(n \log n)$ behavior. The practical speed of our algorithm prompted us to analyze the expected time complexity of our hash-based algorithm: investigating stochastic questions given by random rays on a rectangular grid allows us to predict an $O(n\sqrt{n} \log n)$ expected complexity of our algorithm for n random motorcycles.

Our implementation can handle a set of motorcycles and a set of rigid walls formed by straight-line segments. (If a motorcycle crashes into a wall then it stops, too.) Furthermore, our implementation can cope with simultaneous crashes of motorcycles at the same place, and new motorcycles can be launched at arbitrary points in the course of computation. Thus, our motorcycle-graph algorithm could be used to compute the straight skeleton by means of an approach similar to the one by Cheng and Vigneron [1].

1.3 Definitions

We regard a triple $m = (p, s, t^*) \in \mathbb{R}^2 \times \mathbb{R}^2 \times [0, \infty)$ as a motorcycle, where p denotes the start point, s denotes the speed vector, and t^* denotes the start time. We denote by $R_m(t) := \{p + t's \ : \ t^* \leq t' \leq t\}$ the supporting ray of m until time t and define and $R_m(\infty) := \{p + t's \ : \ t^* \leq t'\}$. The time $T_m(q)$ when m reaches a point $q \in R_m(\infty)$ is given by $T_m(q) := \frac{(q-p)\cdot s}{||s||^2}$.

Let $m_i = (p_i, s_i, t_i^*)$, with $i \in \{1, \ldots, n\}$, be n motorcycles and assume that their start points $p_1, \ldots, p_n$

[*]Work supported by Austrian FWF Grant L367-N15.

[†]Universität Salzburg, FB Computerwissenschaften, A-5020 Salzburg, Austria, `shuber@cosy.sbg.ac.at`

[‡]Universität Salzburg, FB Computerwissenschaften, A-5020 Salzburg, Austria, `held@cosy.sbg.ac.at`

[1]In personal communication with David Eppstein and Siu-Wing Cheng we learned that they also are not aware of implementations of their algorithms.

are pairwise distinct. We consider the following system of conditions for the times $t_1^\dagger, \dots, t_n^\dagger \in [0, \infty]$:

$$\begin{aligned} &\forall i \in \{1, \dots, n\}\ \forall j \in \{1, \dots, n\} \setminus \{i\} : \\ &\qquad \forall p \in R_{m_i}(t_i^\dagger) \cap R_{m_j}(t_j^\dagger) : \\ &\qquad\qquad T_{m_i}(p) \geq T_{m_j}(p) \Rightarrow t_i^\dagger \leq T_{m_i}(p) \quad (1) \end{aligned}$$

Lemma 1 *There always exists a solution vector $(t_1^\dagger, \dots, t_n^\dagger)$ that fulfills Conditions (1). The solution vector is unique if it is requested to be maximal according to lexicographical order, after rearranging all solutions $t_i^\dagger$ in sorted order.*

Proof. If the supporting rays $R_{m_1}(\infty), \dots, R_{m_n}(\infty)$ do not cross, then we can set $t_1^\dagger = \dots = t_n^\dagger := \infty$. So suppose that there are intersections among the supporting rays. We choose an intersection point $q \in \bigcup_{i,j \neq i} R_{m_i}(\infty) \cap R_{m_j}(\infty)$ such that $\max(T_{m_i}(q), T_{m_j}(q))$ is minimized over all intersections. W.l.o.g., we assume that $T_{m_i}(q) \geq T_{m_j}(q)$ and let $t := T_{m_i}(q)$. We can regard m_i as the first motorcycle which crashes; it crashes against m_j at point q in time t. Obviously, setting $t_1^\dagger = \dots = t_n^\dagger := t$ allows us to fulfill Conditions (1). On the other hand, if we choose $t_i^\dagger > t$ then at least one of the conditions of (1) is violated. Hence, we set $t_i^\dagger := t$. Now we repeat our considerations for intersections among supporting rays of motorcycles in $\{m_1, \dots, m_n\} \setminus \{m_i\}$, and interpret the result as the second crash of a motorcycle. We keep going until no intersections among the supporting rays of the remaining motorcycles exist. We set $t_k^\dagger := \infty$ for all remaining motorcycles m_k.

By construction, $t_1^\dagger, \dots, t_n^\dagger$ fulfill Conditions (1). Also due to the construction, the solutions are obtained in sorted order and, thus, are maximal and unique. □

Definition 1 *We call $t_k^\dagger$ (of Conditions (1)) the crashing time of the k-th motorcycle m_k, and we denote by $S_k := R_{m_k}(t_k^\dagger)$ the trace of m_k.*

Definition 2 *We call $\bigcup_{k=1}^n S_k$ the motorcycle graph of the n motorcycles $m_1, \dots, m_n$.*

Corollary 2 *Motorcycle traces do not intersect in their relative interiors.*

Algorithm 1 A simple algorithm for computing motorcycle graphs is obtained by converting the proof of Lem. 1 into an algorithm: we find the crashes of the motorcycles iteratively in chronological order.

We are not aware of an prior formal definition of a motorcycle graph. However, our definition fits to the considerations of prior work. This can be seen by figuring out that Algorithm 1 exactly computes what is described as "motorcycle graph" in prior publications.

In practice it may be requested to consider straight-line segments as rigid walls such that motorcycles crash when they run into a wall. We can easily extend our definition to satisfy this request: in Def. 2, we do not choose $t_k^\dagger \in [0, \infty]$, but $t_k^\dagger \in [0, t_k^\ddagger]$, where $t_i^\ddagger$ denotes the minimal time $T_{m_k}(q)$ for intersection points q of $R_{m_k}(\infty)$ with a wall. If no such intersection exists for m_k then we resort to the old definition by setting $t_k^\ddagger := \infty$.

2 Computing Motorcycle Graphs

2.1 Algorithm

We first discuss the algorithm by Cheng and Vigneron [1], as our algorithm can be interpreted as a practice-minded simplification of their algorithm. Since there are $O(n^2)$ many intersections of the supporting rays of the motorcycle traces, but only $O(n)$ of them realize a crash, Cheng and Vigneron try to reduce the complexity of interactions among the motorcycles. This is done by using so-called $1/\sqrt{n}$-cuttings, which can be interpreted as a very special kind of geometric hashing. A $1/\sqrt{n}$-cutting is a partition of $\mathbb{R}^2$ into a set of simplices; it has the powerful property that no more than $O(\sqrt{n})$ rays intersect a single simplex.

Basically, the algorithm of Cheng and Vigneron is a discrete simulation of the movement of the motorcycles on the cutting. The simulation consists of two types of events: crash events and switch events. The former one indicates a crash of a motorcycle; the later one indicates a switch of a motorcycle from one simplex of the cutting to a neighboring one. In the course of simulation, events are put into a priority-queue, and the algorithm iteratively fetches the earliest event and processes it. However, we are not aware of any implementation of Cheng and Vigneron's algorithm. In any case, implementing the algorithm for the $1/\sqrt{n}$-cutting does not seem to be easy. In our approach, we replace the $1/\sqrt{n}$-cutting of Cheng and Vigneron by a simple regular rectangular hash grid and drop their arrangements.

The input for our algorithm is a set of motorcycles $M = \{m_1, \dots, m_n\}$, as defined in Sec. 1.3, and a set W of line segments representing the walls. We assume that all motorcycles start at distinct points that lie within a unit bounding box. We restrict our computation to a larger copy of the bounding box, which can be imposed easily by adding four walls representing the boundary. We maintain a priority queue[2] Q of pending crash and switch events and a list C of (balanced) binary search trees $C[m]$ for each motorcycle m. As already mentioned, we simulate the movement of the motorcycles, as Cheng and Vigneron do: we use a geometric hash H (consisting of uniform rect-

[2]We may use some sort of minimizing heap, where the priority is the occurrence time of an event.

angular cells) for tracking the motorcycle traces and a geometric hash G for the wall-segments. We use the same $h \times h$ uniform rectangular grid for both H and G, with $h \in \Theta(\sqrt{n})$.

Algorithm 2 The basic algorithm first fills G with all walls from W and invokes `insertMc(`m`)` for every motorcycle $m \in M$. Then, the main loop of the algorithm successively fetches the minimal element e from Q and invokes `handle(`e`)` on it, depending on the type of the event e. The procedures `insertMc()` and `handle()` are described in the sequel.

`insertMc(motorcycle` m`)`: We first add to C an empty binary search tree $C[m]$. Then we insert a switch event e for m to Q, with the time of e set to the start time of m.

`handle(switch event)`: We denote by t, m, c the occurrence time, the motorcycle affected, and the cell that is being entered. At first, we add the next switch event of m to Q, if m leaves c at some point in the future.

Then we get all walls from G that are in cells that intersect c and denote them by *walls*. If m crashes into elements of *walls* then we determine the wall with minimal crashing time and add a respective crash event to Q.

Now we reconstruct the search tree of potential future crash events. First, we clear $C[m]$ and denote by *mcs* all other motorcycles currently associated with c. For every $m' \in mcs$, we check whether the supporting rays of m and m' intersect within c. If they intersect then we denote by q' the potential crash point. If m' reaches q' before m then we add a corresponding crash event of m into m' to $C[m]$. If, on the other hand, m reaches q' before m' does then we denote by e' a potential crash event of m' into m. If e' is not earlier than the earliest event in $C[m']$ then we insert e' into $C[m']$. Otherwise, we update Q: we remove all possibly remaining crash events of m' from Q and re-add them to $C[m']$, and also add e' to Q.

Finally, we add the minimal element of $C[m]$ to Q, if one exists, and associate m with the cell c of H.

`handle(crash event)`: Let t, m, c denote the occurrence time, the motorcycle affected, and the cell of H in which the crash event occurs. First, we mark m as being crashed, clear $C[m]$, and remove the possibly remaining switch event of m in Q. Note that the trace of m ends at the crash point.

Then we clean up the interactions with other motorcycles: let *mcs* be the other motorcycles that are associated with the cell c. For every $m' \in mcs$, we remove from Q the crash event of m' against m, if the crash event has become invalid since it occurs outside of the trace of m. Further, if $C[m']$ contains an invalid crash against m then it is also removed from $C[m']$.

Obviously, Alg. 2 determines the crash events in chronological order. Furthermore, we note that Q contains at no time t a crash event against a motorcycle m at place q, if m already crashed and never reached q. Thus, there are no "stale" crash events in Q.

Let k be the number of motorcycles in a hash cell. Then a crash event is handled in $O(k \log n)$ time[3]. Same holds for switch events. Since there are $O(n)$ crash events and $O(n\sqrt{n})$ switch events, and $k \in O(n)$, we get $O(nkh \log n) \subseteq O(n^2\sqrt{n} \log n)$ as the worst-case complexity. Obviously, the worst case would take place if $\Omega(n)$ motorcycles would cross $\Omega(h)$ hash-cells before crashing. For big data sets this basically means that most motorcycles move parallel to each other in a strip with the thickness of a few hash-cells and no other motorcycles cross the strip before them.

2.2 Expected run-time

As witnessed by our experiments, our implementation achieves an "almost linear" run-time on the vast majority of our data sets. This experimental result motivates a formal analysis of the expected run-time of our algorithm. Studying questions related to random rays within a hash grid allows us to substantiate claims for a good average-case complexity of our algorithm. We summarize our results in the following two theorems.

Theorem 3 *Let S be a unit square covered by a $h \times h$ grid. We distribute n random[4] rays in S. The expectation of the number of rays intersecting any cell of the grid is in $\Theta(\frac{n}{h})$.*

Theorem 4 *Consider n random motorcycles within the unit square S. The expected number of cells intersected by a motorcycle trace is in $\Theta(\sqrt[4]{n})$.*

Proofs related to our stochastic analysis are omitted due to lack of space. However, experimental evidence for the second theorem is provided by our tests: see the plot of the mean trace lengths in Fig. 1. As an immediate consequence, we get that the mean number of switch events per motorcycle is in $\Theta(\sqrt[4]{n})$. Vice versa, in a hash cell there are $\Theta(\sqrt[4]{n})$ motorcycles on average. A combination of these results yields an expected run-time of $O(n\sqrt{n} \log n)$, as claimed.

[3]We can remove an arbitrary element of Q in $O(\log n)$ time, if we maintain a pointer in Q.

[4]Start point and direction angle of each ray are distributed uniformly on $S \times [0, 2\pi)$.

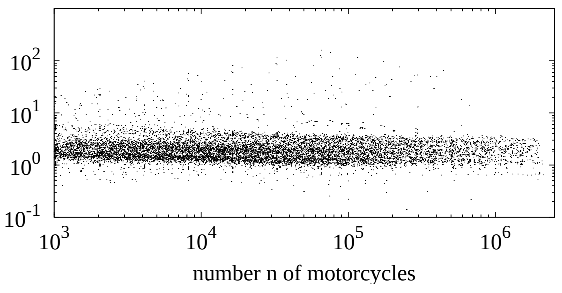


Figure 1: This plot shows for every data set the mean trace length of the motorcycles, multiplied by $\sqrt{n}$, i.e., by the square root of the number of motorcycles.

3 Experimental Results

Our code is called MOCA [5]. It was implemented in C++, based on standard IEEE 754 double-precision floating-point arithmetic. To the best of our knowledge, this is the first implementation of a subquadratic motorcycle-graph algorithm. For this reason, we do not compare our code with other implementations, but content ourselves with a discussion of the performance of MOCA. The tests presented were run on a 32-bit Debian Linux machine, with a 2.66 GHz Core Duo Intel processor, using 4 GB of RAM. For time measurement, we used the C function `getrusage()` and sum up user and system time.

For our tests we obtained motorcycles from straight-line polygonal chains[6]: We generate motorcycles by considering three consecutive vertices v', v'' and v''' in a chain. A motorcycle starts at time 0 and place v'', in direction $\frac{v''-v'}{||v''-v'||} + \frac{v''-v'''}{||v''-v'''||}$, and with speed $\frac{1}{\sin \alpha/2}$, where α is the angle between the vertices v', v'' and v'''. This corresponds to the set-up used by Cheng and Vigneron [1]. We ran MOCA on more than 22 000 polygonal data sets, consisting of synthetic and real-world data. Our real-world data sets — obtained from companies, colleagues, and the web — include polygonal cross-sections of human organs, GIS maps of roads and river networks, polygonal outlines of fonts, and boundaries of work-pieces for NC machining or stereo-lithography, and the like. The synthetic data also contains contrived data, like extremely smooth polygons or highly irregularly distributed vertices.

Figure 2 illustrates the actual run-time on every single data set in a double-logarithmic run-time plot, with the time given in seconds on the y-axis. For a better illustration, the run-times are divided by the number n of motorcycles processed. (To avoid unreliable timings and other idiosyncrasies of small data sets, we only plot results for test runs with at least

[5]MOtorcycle CrAsher.

[6]The polygonal chains may be open or closed and need not be simple. Several chains per test can be handled.

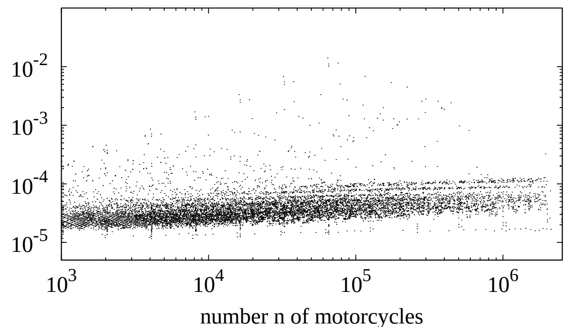


Figure 2: This plot shows the actual run time in seconds on about 22 000 data sets. Depicted are the runtimes divided by the number n of motorcycles.

1000 motorcycles.) A least-squares fit reveals that MOCA processes n motorcycles in $5.05489 \cdot 10^{-6} n \log n$ milliseconds on average on our computer. In our experiments, the polygonal chains were inserted as walls. Anyhow, additional tests demonstrated that inserting or disregarding the polygonal chains has hardly any impact on the run-time.

4 Conclusion

We introduce an easy-to-implement algorithm for computing motorcycle graphs and obtain the surprising result that the combination with geometric hashing makes it very competitive in practice. Extensive practical tests of our C++ on over 22 000 data sets clearly demonstrate the reliability and speed of the C++ implementation of our algorithm, based on standard IEEE 754 floating-point arithmetic: Our implementation processes n motorcycles in $5.05489 \cdot 10^{-6} n \log n$ ms on average on our computer. This experimental result is backed by a stochastic analysis, which leads to $O(n\sqrt{n} \log n)$ as the expected run-time.

References

[1] S.-W. Cheng and A. Vigneron. Motorcycle Graphs and Straight Skeletons. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 156–165, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[2] J. Czyzowicz, I. Rival, and J. Urrutia. Galleries, Light Matchings and Visibility Graphs. In *WADS '89: Proc. of the Workshop on Algorithms and Data Structures*, pages 316–324, London, UK, 1989. Springer-Verlag.

[3] D. Eppstein and J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete Comput. Geom.*, 22(4):569–592, Dec 1999.

[4] D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf. Motorcycle Graphs: Canonical Quad Mesh Partitioning. *Computer Graphics Forum*, 27(5):1477–1486, Sep 2008.

Robust Extraction of the Medial Axes of 3D Objects

Kazutoshi Kan* Kokichi Sugihara†

Abstract

In this paper, we propose a robust algorithm to simplify the medial axes of 3D objects. This method first assigns weights to points on the medial axis in such a way that important points get larger weights, and next simplifies the medial axis by removing points whose weights are smaller than threshold. The resulting simplified medial axis is guaranteed to be homotopically equivalent to the original object for any value of the threshold. This method works well even for constricted objects like hourglasses.

1 Introduction

The Medial Axis (MA) is a basic representation of a geometric shape and provides many useful tools for shape description, image analysis, and shape recognition. However, the MA has an inherent instability that the perturbation of the boundary shape induces large modification in its MA. This is the reason for the difficulty of approximating the MA in a stable manner.

Various methods to simplify the MA are proposed [1, 3, 5]. However for constricted objects like hourglasses, these methods sometimes evaluate the centers of the objects quite lower than their actual importance. In addition to this, their method does not necessarily preserve homotopy or need complicated processes to preserve the homotopy equivalence.

We take a new approach to prune the MA of 3D objects. Our method has the following good properties.

- It prunes branches generated by perturbation of the boundary shape
- It works well for constricted shapes
- It preserves homotopy with a simple process
- As the threshold increases, the simplified MA becomes smaller monotonously

We first focus on the research of Sakai & Sugihara [4]. They proposed a robust method to prune the MAs of 2D objects. It satisfies these properties. In our research, we try to extend their method to 3D objects.

In their method, each point on the MA is associated with a segment of the boundary of the original object, and is assigned the length of the segment as the weight. These weights evaluate the centers of constricted object higher and branches generated by perturbation lower. The weights have a monotone property: as the threshold increases, the simplified MA becomes smaller monotonously. At the same time homotopy is preserved. These excellent properties allow us to prune unnecessary branches effectively.

Our objective is to construct similar monotone weights to the MA of a 3D object. For an analogy to the two dimensional case, we try to define weights using areas of parts of the boundary of a 3D object.

To accomplish our purpose, we use the concept of the Curve Skeleton (CS) [2]. It is a one dimensional object and is homotopy equivalent to its original shape. Our basic idea is as follows: first, we map the shape X to the MA using homotopic function. Next we map the MA to the CS. Then we construct monotone weights for points on the CS; that is easy because it is one dimensional. Then we construct monotone weights for the MA using these weights with desired properties.

2 Instability of the Medial Axis

The MA is expected to be a figure centered in a shape. Especially we call the MA medial surface for a 3D object because it consists of 2-dimensional surfaces. Mathematical definition of the MA is as follows.

Let shape $X \subset \mathbb{R}^n$ be an open set. For any $x \in X$, $\Gamma(x)$ denotes the set of all the nearest points on boundary ∂X from $x \in X$:

$$\Gamma(x) = \{y \in X^c : d(x,y) = d(x,X^c)\},$$

where X^c represents the compliment of X, and d represents the Euclidian distance. Then the MA of the shape X is defined as the set of points in X that have at least two closest points on the boundary ∂X. That is,

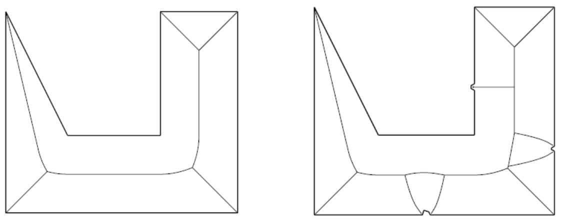
$$\mathrm{MA}[X] = \{x \in X : |\Gamma(x)| \geq 2\}.$$

The MA has instability in the sense that perturbation of the boundary of the shape induces large modification of the MA. For example, Figure 2 shows the MA of a 2D shape in panel (a) and that of a perturbed shape in panel (b). We can observe that small changes of the boundary generate nonessential branches in the MA. In the case of 3D objects, nonessential two dimensional branches are generated.

In applications, input data of a shape has observational and numerical errors as perturbations. Thus the MA obtained directly from the input data includes

*Department of Mathematical Informatics, University of Tokyo, kazutoshi_kan@mist.i.u-tokyo.ac.jp

†Department of Mathematical Informatics, University of Tokyo, sugihara@mist.i.u-tokyo.ac.jp



(a) the MA of a 2D shape (b) the MA of the perturbed shape

Figure 1: instability of the MA

many unnecessary parts and is quite different from the ideal MA which we want to get. This motivates us to prune the MA computed from the scanned data.

3 Curve Skeleton

In this section, we briefly explain the Curve Skeleton (CS) proposed by Dey & Sun [2], and extend it for our purpose. The CS is the one dimensional axis in a 3D object $X \subset \mathbb{R}^3$. In what follows, we assume that the shape X is bounded and its boundary ∂X is smooth and connected.

We first introduce the Medial Geodesic Function (MGF). This function is defined in MA_2, where MA_2 is the set of points which satisfy $|\Gamma(x)| = 2$.

Definition 1 (Medial Geodesic Function) *For a point $x \in \mathrm{MA}_2$, let $\Gamma(x) = \{a_x, b_x\}$. Then, we define $f(x)$ as the geodesic distance on the boundary ∂X between a_x and b_x, and call it the Medial Geodesic Function (*MGF*).*

We want to define the CS as the set of singular points of MGF on MA_2. This definition is adequate if the MA is 2-manifold, but the MA is non-manifold in general and this definition is not sufficient. Hence we extend this definition. In what following, we call a maximum 2-manifold part of the MA as a *sheet*.

We define a divergence of ∇f to extend the definition of the CS. The domain of this definition is $\mathrm{cl}(\mathrm{MA}_2)$.

Definition 2 (Divergence) *For any point $x \in \mathrm{cl}(\mathrm{MA}_2)$, we define $\mathrm{div}(x)|_S$ by*

$$\mathrm{div}(x)|_S = \lim_{|B|\to 0} \frac{1}{|B|} \int_{\partial B} f \cdot \mathbf{n}\, \mathrm{d}s,$$

where S is the sheet containing x, B is an open neighborhood of x constrained in S, and $\mathbf{n}$ is the normal vector on the boundary towards outside of B.

We call $\mathrm{div}(x)|_S$ the divergence of ∇f at x on the sheet S. S can be omitted when $x \in \mathrm{MA}_2$ because the sheet containing x is unique.

We now define the CS using this divergence.

Definition 3 (Curve Skeleton) *Let A be the set of points x belonging to the boundaries of two or more sheets in the* MA *such that the divergence of ∇f at x is negative for all sheets containing x. Let B be the set of points that are not contained in any sheet. Then the* CS *is defined as the closure of $A \cup B$.*

The reason for requiring the closure is to prevent for the CS to become disconnected because of singular points.

Conjecture 1 *The* CS *is homotopy equivalent to X.*

4 Global and Local Weights of the Curve Skeleton and Medial Axis

In this section, we give global weights to points in the MA and the CS. At first, we define a local weight of a subset of the MA as the ratio of the area of the corresponding region to the total area of the boundary. That is, for any subset A of the MA,

$$w(A) = 2 \cdot S(\bigcup_{x\in A} \Gamma(x))/S(\partial X),$$

where $S(Y)$ means the area of surface Y.

Next, we construct a global weight of the CS. We assume the shape X is homotopy equivalent to a single point and the CS is one dimensional object. Hence the CS is a tree.

If we split the CS at a point $x \in \mathrm{CS}$, the CS is divided into two or more subtrees $T_1, \ldots, T_k$ $(k \geq 2)$. The weight of each subtree is defined as the local weight of the union of the associated surface points: that is

$$w\left(\bigcup_{y\in T_i} p(y)\right),$$

where for a point $y \in \mathrm{CS}$, $p(y)$ means the set of points that the flow of ∇f starting at the point arrives at y. We define the global weight of a point $x \in \mathrm{CS}$ as the second largest value of the weights of these subtrees $T_1, \ldots, T_k$. Suppose that the flow starting from $x \in \mathrm{MA}$ reaches a point $y \in \mathrm{CS}$. Then we define a global weight of a point x as the weight of y.

This weight gets larger values at main part of the axis, and smaller values at perturbation-originated branches, because it represents the area of the surface part that generates the associated axis point.

Conjecture 2 *For almost every $x \in$* MA*, the flow of ∇f starting from x reaches a point in the* CS *and the arrival point is unique.*

Conjecture 3 *The global weight of the* MA *has monotone property at almost everywhere: for any threshold, the part of the* MA *that the weight of its point is larger than the threshold is homotopy equivalent to the original* MA *or is empty.*

Conjectures 1, 2 and 3 turn out to be true if we restrict to objects bounded by triangular meshes, and consequently the algorithm proposed in this paper is valid, as will be seen in the next section.

5 Algorithms

Our algorithm takes the 2-manifold surface mesh P as an input shape. It outputs the global weights of the MA and the CS. Our algorithm consists of four steps. In the first step, we find a collection of Voronoi facets as the MA. In the second step, we use the method proposed by Dey & Sun [2] with some modifications to extract the CS. In the third step we assign the ∇f vectors as labels for the facets in MA. In the last step, we give the local and global weights for vertices and facets in the MA and vertices and edges in the CS using labels. We will describe those steps in more details.

5.1 Finding the Medial Axis and Curve Skeleton

Step 1: At first, we generate a Voronoi diagram from the vertices of P and then extract Voronoi facets contained in P. We regard them to be the MA. If they are not homotopy equivalent to a single point, we refine the surface mesh of P by dividing triangles into four subtriangles using new vertices inserted at middle points and try again.

Step 2: We choose edges in the MA as the candidates of CS and calculate ∇f vector for each facet of the MA.

Let a_F and b_F be the two endpoints of the Delaunay edge dual to a facet F in the MA. We find the shortest path $\{a_F, u_1, \dots, u_k, b_F\}$ on the boundary from a_F to b_F. Then we focus on two vectors that come towards the endpoints, i.e., $\mathbf{v}_{a_F} = a_F - u_1$ and $\mathbf{v}_{b_F} = b_F - u_k$. We project the two vectors to the plane containing F and sum them. At last we normalize the vector and regard it as the ∇f vector of the facet F.

Next we find the CS. Let $\mathrm{div}|_F(e)$ be the divergence at an edge e for the facet F, and let $\mathbf{n}$ be the normal vector of e towards inside of F on the plane containing F. Then $\mathrm{div}|_F(e)$ is defined as the inner product of the normal vector $\mathbf{n}$ and the ∇f vector at F:

$$\mathrm{div}|_F(e) = \mathbf{n} \cdot \nabla f(F).$$

We introduce a parameter θ and initialize it to 1. We extract edges in the MA such that the divergence is less than θ for all facets containing the edge. If the set of extracted edges does not contain any loop, we regard it to be a subset of the CS, else decrease the value of θ and try extraction again.

After that, we prune the facets and the edges from the boundary of the MA as long as the remaining part of the MA preserves its homotopy until all the facets are removed. We regard the set of remaining edges to be the CS.

5.2 Label Assignment

Step 3: Suppose that the flow of ∇f starting at a facet F reaches an edge e in the CS. Then we label the facet F with the edge e.

Precisely, consider the flow on the facet F and select an edge e that the divergence is minimum among all the edges of F. Then we choose the next facet of the flow as the facet on the other side of e. If there are more than three facets containing the edge e, we choose the facet that has the largest divergence and regard it to be the next facet F_{next}:

$$F_{\mathrm{next}} = \underset{F' \in \mathrm{N}(e) \setminus F}{\mathrm{argmax}} \; (\mathrm{div}|_{F'}(e)),$$

where $\mathrm{N}(e)$ is the set of facets containing the edge e.

5.3 Calculation of the Global Weights

Step 4: For each vertex v of the MA, we find a Delaunay tetrahedron dual to v, and calculate the ratio of the area of the facet on the boundary of P to the total area of the boundary and double it. We consider it as the local weight of v and denote it as $\mathrm{LVW}(v)$. If any facet of the Delaunay tetrahedron is not on the boundary, the local weight of the vertex is set to be zero.

Next we translate the local weights of vertices to the weights of facets. We equally divide the local weight of a vertex v to facets containing v and accumulate them for all vertices contained in a facet F. Then regard it as the local weight of the facet F and denote it as $\mathrm{LFW}(F)$. This translation preserves the total amount of the local weights.

We now calculate the local weight for the CS edges. We denote the local weight of an edge e of the CS as $\mathrm{LEW}(e)$. Then $\mathrm{LEW}(e)$ is the summation of the local weights of facets labeled with e:

$$\mathrm{LEW}(e) = \sum_{F \in \mathrm{MA},\ \mathrm{label}(F) = e} \mathrm{LFW}(F),$$

where $\mathrm{label}(F)$ is an edge name labeled to F.

For any vertex v of the CS when we split the CS at v, we get subtrees $T_1, \dots, T_k$ ($k \geq 2$). Then we accumulate the $\mathrm{LEW}(e)$ in each subtree and select the second largest value of them as the global weight of v. This is a global weight of a vertex of the CS and we denote it as $\mathrm{GVW}(v)$. Then $\mathrm{GEW}(e)$, the global weight of an edge e of the CS, is defined as the average of GVW of the both endpoints. After that, we assign facets of the MA with the global weights of the associated edges of the CS, and regard it as the global weight of the facet of MA. We denote it as GFW.

The outputs of our algorithm are the global weights of the facets GEW of the MA and the edges of the CS GFW. It is easy to confirm the following proposition from the procedure for computing the global weight.

Proposition 1 *The global weights of the* CS *has a monotone property.*

5.4 Hole Filling

Let the set of edges of the CS and facets of the MA be Y. It is desirable to preserve homotopy of Y while we delete from Y the part of the MA and the CS that the global weight is less than the threshold.

However it can occur that a hole is generated and hence the homotopy changes. In that case, we fill the hole as post-processing to guarantee the homotopy equivalence. For this purpose, we find the loop as the boundary of the hole. The loop cuts out a part from the 2-manifold sheets of the MA or divide them into some parts. Therefore we recover facets inside hole. This operation is rarely necessary in practice.

6 Example

Figure 2 shows an example of the behavior of our algorithm. An input shape "Homer" is shown in Figure 2(a). Figure 2(b) is the initial CS, and Figures 2(c) and 2(d) are pruned ones with the threshold values 0.01 and 0.1, respectively. We can see that the CS shrinks gradually as the threshold increases. Figure 2(e) is the initial MA$\cup$CS, that is, the MA associated with Figure 2(b) and the CS in Figure 2(b). Figures 2(f) and 2(g) are the pruned MA$\cup$CSs associated with Figures 2(c) and 2(d), respectively. We can see that as the threshold increases, the MA $\cup$ CS also shrinks and thus insignificant branches are pruned.

7 Conclusion

We proposed a robust method to simplify the MA of 3D objects. This method prunes the MA by weights which reflect the global significance of the objects, and preserves its homotopy. Especially it works well even for constricted objects.

The basic properties of the CS and MA defined mathematically are still conjectures (i.e., Conjectures 1, 2 and 3), but we could construct our algorithm in such a way that those basic properties are guaranteed for the digital approximation of the CS and MA extracted in the procedure of the algorithm. That is, the output CS and MA are guaranteed to be homotopy equivalent to the input object no matter what value is chosen as threshold for pruning. To prove the conjectures in a mathematical framework is one of the problems for future.

We can extend our method for the object that are not homotopy equivalent to a single point. However we omit details because of the space limitation.

Acknowledgments

We would like to acknowledge CGAL library we used for our program. We thank AIM@SHAPE for pro-

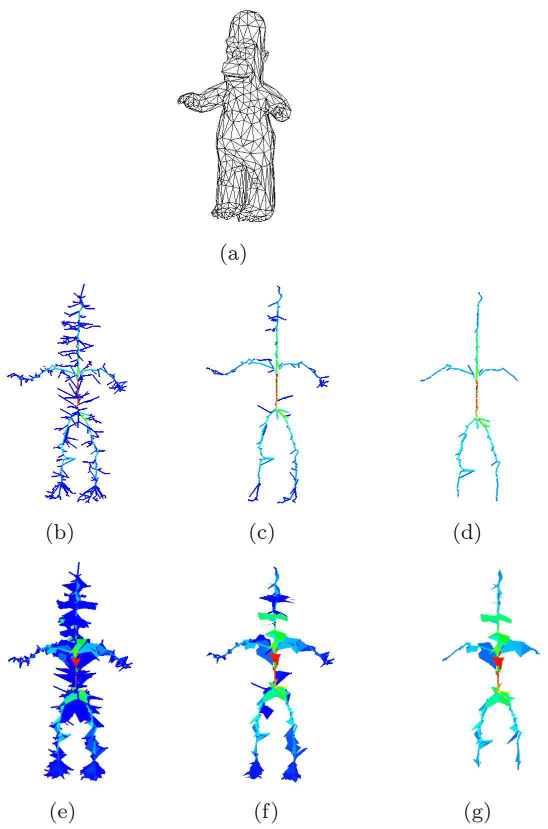


Figure 2: Results of our algorithm

viding 3D models and Tamal. K. Dey [2] of Ohio State University for making his software available. We are deeply grateful to Shinji Imahori of University of Tokyo for his valuable advices. This work is supposed by the Grant-in-Aid for Scientific Research (B), No. 20360044 of the Japanese Society for Promotion of Science.

References

[1] D. Attali and A. Montanvert: Computing and simplifying 2D and 3D continuous skeletons. *Computer Vision and Image Understanding*, 1997, Vol. 67, 261–273.

[2] T. K. Dey and J. Sun: Defining and computing curve-skeletons with medial geodesic function. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing*, 2006, 143–152.

[3] T. K. Dey and W. Zhao: Approximating the medial axis from the Voronoi diagram with a convergence guarantee. *Algorithmica*, 2003, Vol. 38(1), 179–200.

[4] H. Sakai and K. Sugihara: Stable and topology-preserving extraction of medial axes. In *Proceedings of the 3rd International Symposium on Voronoi Diagrams in science and engineering*, 2006, 40–47.

[5] A. Sud, M. Foskey and D. Monocha: Homotopy-preserving medial axis simplification. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, 2005, 39–50.

Continued Work on the Computation of an Exact Arrangement of Quadrics

Michael Hemmer* Sebastian Limbach* Elmar Schömer†

Abstract

We present an exact and complete algorithm for the computation of all edge cycles bounding the faces of a three-dimensional arrangement of algebraic surfaces of degree two (quadrics). The algorithm is based on the implementation by Hemmer et al. [9] which computes the adjacency graph of the arrangement. However, this graph is just the set of vertices and their connectivity along the edges. We enhance each vertex of the graph by a description of its local neighborhood in a so-called environment map, which finally enables the initialization of all edge cycles in the arrangement. This is a major step towards the computation of the full arrangement. The implementation is complete up to a few special cases, and covers several variants in order to compute the environment map.

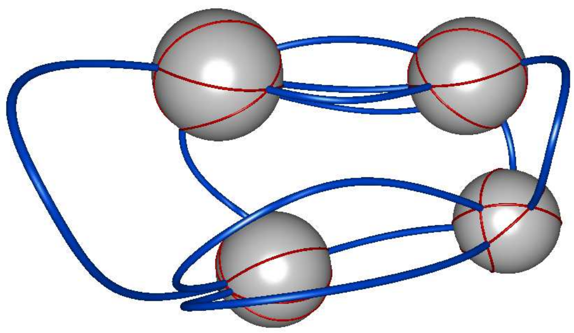


Figure 1: Four environment sphere maps and the corresponding edges of the 3-dimensional arrangement.

1 Introduction

We aim for the computation of the three-dimensional arrangement $\mathcal{A}(\mathcal{Q})$ induced by a given set $\mathcal{Q}$ of quadric surfaces, or quadrics for short. A *quadric* Q_f is given by a trivariate polynomial $f \in \mathbb{Q}[x, y, z]$ of degree 2. Quadrics cover a couple of common surfaces such as spheres, ellipsoids, cones, cylinders, hyperboloids, planes, and double planes. The set of input surfaces may also cover simple rational planes. The *arrangement* $\mathcal{A}(\mathcal{Q})$ is the decomposition of $\mathbb{R}^3$ by the surfaces into cells of dimension 0 (vertices), 1 (edges), 2 (faces) and 3 (volumes) [8]. Arrangements are ubiquitous in computational geometry and can be applied to many problems, for instance, they can serve as a fundamental first step for CSG operations. However, existing inexact, floating point based implementations suffer from rounding errors which can lead to ruinous effects. In contrast, existing exact and complete methods are often not efficient enough for actual application (e.g. [3]). Our goal is to close this gap by following an approach that is exact and complete by design, on the one hand, and efficient enough to be used in practice, on the other hand.

On this way a major goal was recently reached: Hemmer et al. [4, 9] presented an implementation for the computation of the adjacency graph $\mathcal{G}(\mathcal{Q})$. The approach is based on Dupont et al. [5, 10], which provides an exact parameterization of the appearing intersection curves. With these parameterizations, the approach represents the intersection points of quadrics by the exact parameter values with respect to the intersection curves they lie on. Therefore, it is possible to sort the points along the curve, which enables the initialization of the adjacency graph.

Obviously, the adjacency graph $\mathcal{G}(\mathcal{Q})$ does not contain enough information in order to represent the full arrangement as it just stores all vertices and edges of the arrangement $\mathcal{A}(\mathcal{Q})$. The missing parts are the two and three dimensional cells, that is, the faces and volumes of the arrangement and, in particular, their incidences at each vertex. Therefore, we investigate the local neighborhood of each vertex. It is represented by a two dimensional arrangement which is conceptually embedded on a sphere. We call this the *environment map*[1] of a vertex.

A vertex on the sphere corresponds to an incoming edge of $\mathcal{G}(\mathcal{Q})$ and a halfedge on the sphere corresponds to an incident halfface of $\mathcal{A}(\mathcal{Q})$. That is, each face of $\mathcal{A}(\mathcal{Q})$ is bounded by a sequence of halfedges whereas each of these halfedges can be identified with a halfedge of an environment map. This is similar to the existing approach for the arrangement of planes in [6, 7].

*Max-Planck-Institut für Informatik, 66123 Saarbrücken, {hemmer,slimbach}@mpi-inf.mpg.de

†Institut für Informatik, Johannes Gutenberg-Universität Mainz, schoemer@uni-mainz.de

[1]Here, the term "environment map" is not related to its usage in the context of computer graphics.

A more detailed discussion of the environment map is given in Section 2. With these details, Section 3 shows how the edge-cycles in the three dimensional arrangement are initialized. Section 4 presents the available implementations, which are compared in Section 5.

The algorithm is implemented within the context of the EXACUS[2] project.

2 The environment map

Let v be a vertex of $\mathcal{A}(\mathcal{Q})$ and $\mathcal{Q}_v \subseteq \mathcal{Q}$ be the set of quadrics v lies on. Obviously, the quadrics in $\mathcal{Q}_v$ are the only quadrics that influence the local neighborhood. In order to obtain a representation of this local neighborhood the idea is to intersect $\mathcal{Q}_v$ with a sufficiently small sphere $\mathcal{S}$ centered at v. This results in a two-dimensional arrangement $\mathcal{A}(\mathcal{C})$ induced by the intersection curves $\mathcal{C} = \{\mathcal{C}_i = \mathcal{Q}_i \cap \mathcal{S} \mid \mathcal{Q}_i \in \mathcal{Q}_v\}$. The arrangement $\mathcal{A}(\mathcal{C})$ consists of cells of dimension 0, 1 and 2 on the surface of $\mathcal{S}$.

Obviously, there exists an r_0 such that every arrangement on a sphere $\mathcal{S}_r$ with radius $r \leq r_0$ has the same topology as the arrangement on $\mathcal{S}_{r_0}$. The fundamental question is how to detect that the sphere is actually small enough. Obviously, the arrangement does not change once the sphere is small enough with respect to every single quadric, every pair of quadrics and every triple of quadrics out of $\mathcal{Q}_v$. The following three constrains ensure this.

1. If v is the apex of a cone, or v lies in the common line of intersecting planes, the topology of the intersection of this surface with $\mathcal{S}$ will not change at all. For any other quadric, the intersection with the interior of $\mathcal{S}$ must be homeomorphic to a disk, that is, the quadric induces exactly one intersection curve homeomorphic to a circle on the sphere.

2. The intersection of every pair of quadrics (intersection curves) of $\mathcal{Q}_v$ intersected with the interior of $\mathcal{S}$ must be homeomorphic to a line segment (or two intersecting line segments, in case of a nodal quartic with v at its nodal point).

3. The intersection of triples of quadrics (vertices) of $\mathcal{Q}_v$ intersected with the interior of $\mathcal{S}$ of correct size must be v itself.

Note that the bounding object does not need to be a sphere, for instance, it could be replaced by a box or an even simpler bounding object. Therefore, we decided to follow the generic programming paradigm [1] using the C++ template mechanism and kept the code generic in the way the local neighborhood is constructed and represented. We can instantiate the code by any type that fulfills a certain set of requirements. Such a set of of requirements is called a *concept*. A type that fulfills all these requirements is called a *model* of the concept. More precisely, we introduced the concept `EnvironmentMap_2`, which we briefly discuss next.

Given a vertex v a model of `EnvironmentMap_2` is supposed to represent the local neighborhood of v by a two dimension arrangement that is conceptually embedded on a sphere. The arrangement is stored in a DCEL data structure consisting of *halfedges*, *vertices* and *faces*. Since a vertex on the sphere corresponds to an incoming edge of $\mathcal{G}(\mathcal{Q})$, the model must identify each vertex with the corresponding outgoing edge of $\mathcal{G}(\mathcal{Q})$. Similarly, every halfedge must be identified with the corresponding quadric. This is complemented by the so-called *alignment flag*, which is used to associate each halfedge with either the inside or the outside of the corresponding quadric. These informations are needed later on, since they allow the identification of the next edge in an 3D-edge cycle. For more details, which are in particular relevant in case of degenerated quadrics (e.g. two intersection planes), we refer to [11].

Note that we actually do not enforce that the local neighborhood is constructed via the intersection with a concrete bounding object. We will use this fact in Section 4, which provides a model that computes the local neighborhood based on the tangent planes of all quadrics in $\mathcal{Q}_v$ at v. This will turn out to be very efficient but it is not applicable in all cases. Therefore, we defined the additional concept `EnvironmentMapFilter_2`. It has the same requirements as `EnvironmentMap_2` and in addition it requires a static function that takes the vertex as input and returns a boolean flag indicating whether the filter is applicable or not.

3 Initializing the edge cycles

The environment maps provide us with all information the algorithm needs for traversing an edge cycle bounding a halfface[3].

We start at an arbitrary halfedge of the environment map of any vertex. The associated surface and the indication whether the halfedge belongs to the inside or the outside of the surface (the alignment flag) correspond uniquely to one of the halffaces adjacent to the halfedge. From the endpoint of our halfedge we travel along the associated edge of $\mathcal{G}(\mathcal{Q})$, reaching a new vertex on the next environment map. We then pick the next outgoing halfedge by comparing the associated surface and the alignment flag and continue

[2] "Efficient and exact algorithms for curves and surfaces", `http://www.mpi-inf.mpg.de/projects/EXACUS/`.

[3] In analogy to the DCEL of our environment maps, we divide each face into two halffaces.

until we reach our starting edge again. The creation of all edge cycles is implemented accordingly.

4 Environment map models

Our implementation consists of two different approaches for the computation of an environment map:

Sphere map model: The first intuitive idea for constructing the environment map of a vertex v is the actual computation of an arrangement on a spherical surface. This results in an arrangement $\mathcal{A}(\mathcal{C})$ of intersection curves of degree up to four. Berberich et al. [2] developed a general framework for sweeping a set of curves embedded on a 2D-parametric surface. An implementation is available in the `Arrangement_on_surface_2` package of CGAL[4]. The sphere map model uses the `Arrangement_on_surface_with_history_2` (AoS) class of this `Arrangement_on_surface_2` package for the computation of the two-dimensional arrangement $\mathcal{A}(\mathcal{C})$ together with a construction history. It is a model of the `EnvironmentMap_2` concept.

The AoS class is capable of constructing the arrangement of intersection curves induced by quadric surfaces on a given base surface. So it remains to construct and to initialize a sphere of the correct size as a base surface, to convert the arrangement obtained from the AoS class into our own DCEL representation, to connect all vertices and halfedges of $\mathcal{A}(\mathcal{C})$ with the corresponding edges and surfaces from the adjacency graph and finally to initialize the alignment flag.

For the matching of the vertices of $\mathcal{A}(\mathcal{C})$ with the outgoing edges at v, we intersect the edges with $\mathcal{S}$ to obtain a set of intersection points on each edge. We then use a matching algorithm based on so-called bigfloat interval (BFI) arithmetic for matching approximations of the positions of the vertices with approximations of the positions of the intersection points. A bigfloat is a multi precision floating point number, that is, the bitsize of its representation is only limited by the available memory. A BFI is an interval number type which uses bigfloats as interval borders. See [9, 11] for more details about the application of BFIs in our algorithms.

For the matching of the halfedges with the corresponding quadrics we only have to look at the construction history of the AoS class. It remains to initialize the alignment flag on each halfedge. For this flag we usually take the midpoint of the successor of the halfedge and check if the point is inside or outside the quadric. Alternatively, if the next halfedge belongs to the quadric we want to check, we take the midpoint of the successor of the twin halfedge and negate the result. If we are in the rare situation that both successors are part of the quadric, we go the other way around: We take any point on $\mathcal{S}$ that is not part of the quadric, compute whether it is at the inside or the outside of it, and check if the point is inside the face bounded by the halfedge, or in any face which is a hole inside the face bounded by the halfedge.

Nef filter model: The computation of an arrangement on a concrete sphere can be a very costly and complex task. At the same time, such arrangements can get simpler if we consider an infinitesimally small sphere. In such a case, the surfaces of all general quadrics are equivalent to their tangential planes at the position of v. To avoid the high algebraic degrees occurring in an exact representation of the normal vectors of these tangential planes, we approximate all vectors using sufficiently precise BFI approximations. The arrangement $\mathcal{A}(\mathcal{C})$ exists and is unambiguous as long as all tangential planes exist and all triples of normal vectors of the involved tangential planes are linearly independent. Therefore, the drawback of this approach is that it can not be applied in general. But the overall performance benefits from the application of such a model as a filter.

The Nef filter model is implemented using the `Nef_polyhedron_S2` class from the `Nef_S2` package of CGAL. This class is capable of representing two-dimensional Nef polyhedrons (arrangements of half-spaces bounded by lines in the plane), embedded on a sphere [6, 7]. Our implementation is a model of the `EnvironmentMapFilter_2` concept.

We first construct the sufficiently precise BFI approximations of all normal vectors of the tangential planes. Afterwards, we can computed the two-dimensional arrangement $\mathcal{A}(\mathcal{C})$ on $\mathcal{S}$ using the `Nef_polyhedron_S2` class. Next, we convert the arrangement into our own DCEL representation. For each new halfedge, we determine the corresponding surface, as well as the value of the alignment flag. We obtain the corresponding surface the halfedge is embedded in by matching the normal vector of the plane induced by the origin and the halfedge with the normal vectors of all tangential planes of the surfaces. Whether the normal vectors have the same or opposite direction indicates the belonging of the halfedge to the inside or the outside of the corresponding surface.

For each vertex of $\mathcal{A}(\mathcal{C})$, we identify the intersection curve the vertex is associated with, and we determine the corresponding edge of $\mathcal{G}(\mathcal{Q})$. We obtain the supporting curve by looking at the two associated surfaces of the halfedges adjacent to the vertex of $\mathcal{A}(\mathcal{C})$. Finally, we determine the corresponding edge we associate with each vertex by matching the direction from v to the vertex with the tangent vector of the supporting curve at v.

[4] "Computational Geometry Algorithms Library", `http://www.cgal.org`.

5 Benchmarks

We compared the runtime of our models by a series of benchmarks. First, we directly compared the performance of our sphere map implementation to the implementation of the Nef filter model on a set of quadrics with random coefficients. The results of this benchmark are presented in table 1.

$\|\mathcal{Q}\|$	sphere map $[s]$	filter $[s]$	adja. graph $[s]$
3	1.3986	0.204885	0.099786
4	18.2372	0.786321	0.244173
5	86.3439	1.68017	0.512816
6	237.234	1.74667	0.337523
7	352.006	2.91448	0.535007
8	580.153	4.79391	0.797845
9	853.958	7.06226	1.14679
10	1168.73	8.6739	1.50417

Table 1: Comparison of the sphere map model and the Nef filter model on random quadrics.

In such a random situation, the probability of having degenerated environments at vertices is practically zero. Therefore, we can always apply the faster Nef filter implementation. One observes that the gain in runtime for the Nef filter potentiates with respect to the increasing number of vertices of the three dimensional arrangement.

$\|\mathcal{Q}\|$	sphere map $[s]$	filter $[s]$	adja. graph $[s]$
3	60.2265	5.62893	1.3789
4	165.422	19.6249	5.02124
5	404.945	38.7777	12.8201
6	971.758	73.1718	25.1551

Table 2: Comparison of the sphere map model and the filter model on an arrangement with one degenerated vertex environment.

For the second benchmark, we compared the performance of the general sphere map model implementation with a filtered model implementation, that decides whether to apply the sphere map or the Nef filtered implementation for each vertex. We tested the implementation on an increasing number of quadrics that all intersect tangentially in one vertex. At this vertex, the filter cannot apply the Nef filter and has to fall back to the slower general model. Table 2 shows the results of the second benchmark. As in the previous benchmark, the sphere map implementation performs worse than the filtered version. The time to take the decision whether or not to apply the filter and the overhead of initializing the edge cycles is negligible compared to the runtimes of the adjacency graph and the environment map computation.

6 Conclusion and outlook

We presented an algorithm, a concept and two different implementations for the computation of all edge cycles bounding the faces of the three-dimensional arrangement of a set of quadrics. For the complete initialization of the faces, however, we still have to detect the holes of each face, that is, information about the nesting of the faces. For that, the idea is to construct intersection curves on the quadrics and to intersect them with the other quadrics of the arrangement. The order of the intersections provides the required information about the nesting of the faces.

References

[1] M. H. Austern. *Generic Programming and the STL.* Addison-Wesley, 1998.

[2] E. Berberich, E. Fogel, D. Halperin, K. Mehlhorn, and R. Wein. Sweeping and maintaining two-dimensional arrangements on surfaces: A first step. In *Proc. ESA 2007*, LNCS, pages 645–656, Eilat, Israel, 2007. Springer.

[3] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183, 1975.

[4] L. Dupont, M. Hemmer, S. Petitjean, and E. Schömer. Complete, exact and efficient implementation for computing the adjacency graph of an arrangment of quadrics. In *Proc. ESA 2007*, LNCS, pages 633–644, Eilat, Israel, 2007. Springer.

[5] L. Dupont, D. Lazard, S. Lazard, and S. Petitjean. Near-optimal parameterization of the intersection of quadrics: I+II+III. *Journal of Symbolic Computation*, 2008.

[6] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3D selective Nef complexes. In *Proc. ESA 2003*, LNCS, pages 654–666, Budapest, Hungary, 2003. Springer.

[7] P. Hachenberger. *Boolean Operations on 3D Selective Nef Complexes.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

[8] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. Chapman & Hall/CRC, 2nd edition, 2004.

[9] M. Hemmer. *Exact Computation of the Adjacency Graph of an Arrangement of Quadrics.* PhD thesis, Johannes Gutenberg-Universität, Mainz, Germany, 2008.

[10] S. Lazard, L. M. Peñaranda, and S. Petitjean. Intersecting quadrics : An efficient and exact implementation. In *Proc. 20th ACM Symposium on Computational Geometry*, Brooklyn, NY, 2004.

[11] S. Limbach. Continued Work on the Computation of an Exact Arrangement of Quadrics. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2008.

Exact Construction of Minimum-Width Annulus of Disks in the Plane[*]

Ophir Setter[†] Dan Halperin[†]

Abstract

The construction of a minimum-width annulus of a set of objects in the plane has useful applications in diverse fields, such as tolerancing metrology and facility location. We present a novel implementation of an algorithm for obtaining a minimum-width annulus containing a given set of disks in the plane, in case one exists. The algorithm extends previously known methods for constructing minimum-width annuli of sets of points. The algorithm for disks requires the construction of two Voronoi diagrams of different types, one of which we call the "farthest-point farthest-site" Voronoi diagram and appears not to have been investigated before. The vertices of the overlay of these two diagrams are candidates for the annulus' center. The implementation employs an asymptotically near-optimal randomized divide-and-conquer algorithm for constructing two-dimensional Voronoi diagrams. Our software utilizes components from CGAL, the Computational Geometry Algorithms Library, and follows the exact computation paradigm. We do not assume general position. Namely, we handle degenerate input and produce exact results.

1 Introduction

An *annulus* is the bounded area between two concentric circles. The width of an annulus is the difference between the radii of its outer and inner bounding circles. Given a set of objects in the plane, the objective is to find a minimum-width annulus containing those objects. Figure 1(c) shows such an annulus for a set of disks. Constructing a minimum-width annulus has applications in various fields including tolerancing metrology and facility location [11, 19].

A minimum-width annulus does not always exist. If the width of the set of objects[1] is smaller than the width of any containing annulus, then there is no minimum-width annulus.

In the case of point sets, a minimum-width annulus must have (at least) two points on each of its outer and inner circles [15]. Hence, the center of a minimum-width annulus must lie on an intersection point of the nearest-neighbor and the farthest-neighbor Voronoi diagrams of the points. Using this observation, an algorithm for finding a minimum-width annulus of planar points was developed [5, 16].

Similar methods were used to solve different variations of the problem, such as finding a minimum-width annulus of point sets with different constraints on its radii (e.g., fixed inner radius) [3], and finding a minimum-width annulus bounding a polygon [10]. For some special cases there are specific deterministic sub-quadratic algorithms [4, 8, 17].

Agarwal and Sharir introduced the most efficient (randomized) algorithm to date for constructing a minimum-width annulus of planar points, which achieves an expected running time of $O(n^{3/2+\varepsilon})$ [1].

2 Solving the Problem for Disks

Given a set of objects O in the plane (also called *Voronoi sites*) and a distance function ρ, the *nearest-neighbor Voronoi diagram* of O with respect to ρ is the partition of the plane into maximally connected cells, where each cell consists of points that are closer to one particular site (or a set of sites) than to any other site. The *bisector* of two Voronoi sites is the locus of all points that have an equal distance to both sites. A similar definition is used to define the *farthest-neighbor Voronoi diagram*.

Recall that in the case of point sets, a minimum-width annulus can be found by overlaying the nearest-neighbor and farthest-neighbor Voronoi diagrams of the points. We show that a similar approach applies to the case of sets of disks, but the relevant diagrams require more careful definitions.

Instead of constructing the nearest-neighbor Voronoi diagram of the points we construct the *additively-weighted Voronoi diagram* of the disks, also known as the *Apollonius diagram* [14].

The Apollonius diagram is the Voronoi diagram defined for disks with respect to the following distance function ρ. For a point p and a disk D with a center c and radius r, we define $\rho(p, D) = ||p - c|| - r$. The distance between a point outside a disk and the disk is the Euclidean distance. Apollonius bisectors, which compose the diagram, are branches of hyperbolas.

Instead of constructing a farthest-neighbor Voronoi diagram of points we construct a different diagram,

[*]This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

[†]School of Computer Science, Tel-Aviv University, 69978, Israel. {ophirset,danha}@post.tau.ac.il

[1]The width of a set is defined to be the width of the thinest strip (i.e., the area bounded between two parallel lines) containing it.

which requires the definition of another distance function. Consider the following farthest-point distance function from a point $p \in \mathbb{R}^2$ to a set of points $S \subset \mathbb{R}^2$:

$$\rho(p, S) = \sup_{x \in S} ||p - x||,$$

which measures the farthest distance from the point p to the set S. Consider the farthest-neighbor Voronoi diagram with respect to this distance function. We call this diagram the "farthest-point farthest-site" (FPFS) Voronoi diagram. The distance function ρ becomes the Euclidean distance when the set S consists of a single point. However, this is not the case when the set S is, say, a disk in the plane.

The following lemma characterizes the FPFS Voronoi diagram of disks in the plane, showing that the bisectors induced by its sites are hyperbolic arcs.

Lemma 1 *The bisector of two disks in the plane induced by the farthest-point distance function is one branch of a hyperbola, and is identical to the Apollonius bisector of the disks with swapped radii.*

Proof. Let $(c_A, r_A), (c_B, r_B)$ be two disks in the plane with respective centers c_A, c_B and radii r_A, r_B. Their farthest-point distance bisector is the zero set of the equation $||x - c_A|| + r_A = ||x - c_B|| + r_B$, which is the same as the zero set of $||x - c_A|| - r_B = ||x - c_B|| - r_A$. The latter equation describes the Apollonius bisector of $(c_A, r_B), (c_B, r_A)$. $\square$

We now prove that there is a minimum-width annulus (in case one exists) whose center is a vertex of the overlay of the Apollonius diagram and the FPFS Voronoi diagram of the disks.

Let $\mathcal{D} = \{d_1, \ldots, d_n\}$ be a collection of disks in the plane, such that for all i, $d_i \not\subseteq \bigcup_{j \neq i} d_j$. For simplicity of exposition, we assume here that $n \geq 3$; the case of $n < 3$ is simple to handle. Let $\mathcal{I}_N, \mathcal{O}_N \subseteq \mathcal{D}$ denote the set of disks that touch the inner and outer circles of a bounding annulus N, respectively. We show that there is a minimum-width annulus whose circles intersect the disks of $\mathcal{D}$ in at least 4 points.

Theorem 2 *If there is a minimum-width annulus containing $\mathcal{D}$, then there is a minimum-width annulus N such that $|\mathcal{I}_N| + |\mathcal{O}_N| \geq 4$.*

We omit the detailed proof in this extended abstract. Each minimum-width annulus must touch the disks of $\mathcal{D}$ in at least two points, as we can shrink $\mathcal{O}_N$ and expand $\mathcal{I}_N$ until each of them touches a disk. In the case where N does not touch the disks of $\mathcal{D}$ in at least 4 points, we can move N and obtain a smaller width annulus.

Applying Theorem 2, we distinguish between three cases for a possible location of the center of a minimum-width annulus:

1. $|\mathcal{I}_N| \geq 3$ and $|\mathcal{O}_N| = 1$ – the center coincides with a vertex of the Apollonius diagram.
2. $|\mathcal{I}_N| = 1$ and $|\mathcal{O}_N| \geq 3$ – the center coincides with a vertex of the FPFS Voronoi diagram.
3. $|\mathcal{I}_N| \geq 2$ and $|\mathcal{O}_N| \geq 2$ – the center lies on an intersection point of the Apollonius diagram and the FPFS Voronoi diagram.

We therefore construct each of the diagrams and overlay them. For each vertex of the overlay, we retrieve four relevant disks (either three touching the inner circle and the one touching the outer circle, in case 1, or three touching the outer circle and the one touching the inner circle, in case 2, or two pairs of disks touching respectively the inner and outer circles), and compute the width of the resulting annulus. We output the annulus of the smallest width. Figure 1 illustrates the algorithm for computing a minimum-width annulus of a set of disks and a highly degenerate input, which is handled properly by our implementation.

The FPFS Voronoi diagram can be defined as a farthest site abstract Voronoi diagram [12]. Hence, FPFS Voronoi diagrams are of linear complexity in the size of the input (as are Apollonius diagrams).

3 Constructing General Voronoi Diagrams with CGAL

As described in Section 2 above, the process of constructing a minimum-width annulus bounding a set of disks mainly comprises three geometric operations: (i) construction of the Apollonius diagram, (ii) construction of the FPFS Voronoi diagram, and (iii) overlay of the two diagrams. This section describes some of the software components which our implementation is based on, their various ramifications on the algorithm, and how they work in synergy to yield an exact and robust implementation, which can handle input that is not necessarily in general position and produce results of arbitrary precision.

The connection between Voronoi diagrams and envelopes is long-known [6], and yields a useful approach for constructing various types of Voronoi diagrams. CGAL,[2] the Computational Geometry Algorithms Library, contains a robust and efficient implementation of a divide-and-conquer algorithm for constructing envelopes of general surfaces in 3-space [13]. This implementation was employed to yield a general framework for constructing two-dimensional Voronoi diagrams [9].

Like most algorithms and data-structures in CGAL, the framework follows the generic programming paradigm, and is independent of the type of input sites. The generic implementation is parameterized with a *traits* class that must provide all required types and methods to construct and handle bisector curves

[2]`http://www.cgal.org`

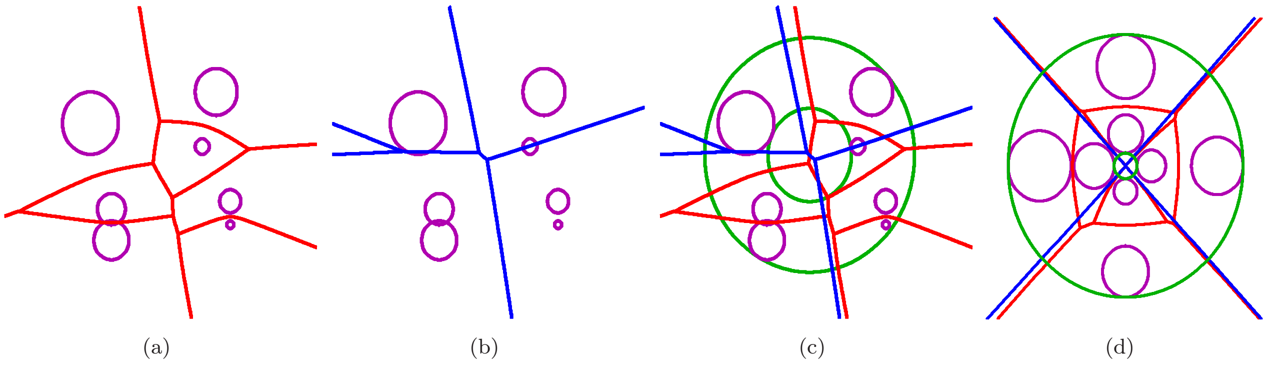


Figure 1: Constructing a minimum-width annulus of a set of disks. (a) The Apollonius diagram of the set of disks. (b) The FPFS Voronoi diagram of the set of disks. (c) A minimum-width annulus of the set of disks. The center of the annulus is a vertex in the overlay of the Apollonius and the FPFS Voronoi diagrams. (d) A highly degenerate scenario for constructing a minimum-width annulus of a set of disks.

(e.g., intersecting bisector curves, comparing the y-coordinates of a point and its vertical projection on a curve), and to answer proximity queries (see Section 4).

The decision to use the aforementioned framework to construct both Voronoi diagrams required in the minimum-width annulus algorithm has two major advantages. First, the framework is bisector-based; namely, the main basic operation it requires is the construction of bisectors of Voronoi sites. This simplifies the implementation of the diagrams as both have the same bisector type (Lemma 1). Second, the resulting diagrams are represented as CGAL arrangements, which can be passed as input to consecutive operations supported by CGAL. One of those operations is a sweep-based overlay [18] that is used to carry out the next step of our algorithm.

A straightforward application of the divide-and-conquer approach for Voronoi diagrams yields algorithms with worst-case running time of $O(n^{2+\varepsilon})$ even for diagrams of linear complexity. Through randomization, it has been shown that the expected running time is lower:

Theorem 3 [9] *For a specific type of two-dimensional Voronoi diagrams, so that the worst-case complexity of the diagram of any set of at most n sites is $O(n)$, the divide-and-conquer envelope algorithm computes it in expected $O(n \log^2 n)$ time.*

In our case, applying Theorem 3 yields an expected construction time of $O(n \log^2 n)$ for both the Apollonius and the FPFS Voronoi diagrams using the divide-and-conquer algorithm. Overlaying the two diagrams with the sweep-based algorithm has $O((n + k) \log n)$ worst-case time complexity where k is the number of intersections between the diagrams. The total expected running time of the algorithm is therefore $O(n \log^2 n + k \log n)$, where k can be $\Theta(n^2)$.

Though the worst-case complexity of the algorithm is super-quadratic, it is reasonable to assume that the expected time complexity is, in many cases, smaller. Indeed, we may have $\Theta(n^2)$ intersections between the two diagrams, but, though not proven for disks, for random point sets it is known that the expected number of intersections between the farthest and the nearest Voronoi diagrams is linear [2].

4 Implementation Details

Following the description of the framework for Voronoi diagrams, we have implemented two traits classes for constructing Apollonius diagrams and FPFS Voronoi diagrams named `Algebraic_apollonius_traits_2` and `Algebraic_farthest_point_farthest_site_traits_2`, respectively.

Both traits classes are based on the algebraic plane curves traits for the arrangement package [7], which provides all the algebraic functionality needed to handle algebraic bisector curves in a robust and exact manner. The remaining functors required by the framework are functors for constructing the bisectors of two disks (which consist of selecting the correct branch of a hyperbola) and functors to answer different proximity queries.

Each required proximity predicate is given a set of points P in the plane (e.g., an edge) and two Voronoi sites, and should indicate which of the sites is closer to P. All required proximity predicates are implemented similarly as follows: we construct a point p inside P using the algebraic infrastructure, and then answer the proximity query by comparing the Apollonius (or the farthest-point) distances from p to the sites. We expedite the comparisons here, as well as the comparisons of widths of annuli later, by using rational interval arithmetic, alleviating the need for the heavier exact algebraic machinery wherever possible.

As both the Apollonius and the FPFS Voronoi diagrams have the same bisectors type and similar distance functions (Lemma 1), both `Algebraic_apollonius_traits_2` and `Algebraic_farthest_point_farthest_site_traits_2` inherit from the same base class, and define only a distance function and a functor for bisectors' construction (`Algebraic_farthest_point_farthest_site_traits_2` swaps the radii of the disks and then calls the same function as `Algebraic_apollonius_traits_2`), maximizing code reuse.

We use the generic overlay function from CGAL's arrangement package to overlay the Apollonius diagram and the FPFS Voronoi diagram. The overlay function is parameterized with an overlay-traits class, which updates the resulting arrangement's features based on data associated with the input arrangements' features [18]. We have created a specially-tailored overlay-traits class for updating the features of the overlay with sites from both diagrams.

Disks	Time	V	E	F
50	10.741	126	213	88
100	26.994	238	395	158
200	62.968	416	659	244
500	185.244	775	1174	400
1000	405.405	1242	1894	653

On the left you can see the time consumption (in seconds) of the algorithm execution on different input sizes, as well as the size of the final overlay (the vertices of which are the candidates for the center of the annulus). The experiments were carried out on an Intel® Core$^{\text{TM}}$2 Duo 2.00GHz processor with 1GB memory.

5 Future Work

It would be interesting to further investigate the farthest-point farthest-site Voronoi diagram, presented above, for disk sites in particular and for other objects in general, and find additional applications for this type of diagrams.

Another direction to pursue is the application of this approach to objects other than points [5, 16], polygons [10], or disks, as described in this paper.

Acknowledgments

The authors are grateful to all those who have supported this work by advice, discussion, or fruitful cooperation. Special thanks are given to Efi Fogel, Eric Berberich, and Barak Raveh.

References

[1] P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Disc. Comput. Geom.*, 16:317–337, 1996.

[2] P. Bose and L. Devroye. Intersections with random geometric objects. *Comput. Geom. Theory Appl.*, 10(3):139–154, June 1998.

[3] M. de Berg, P. Bose, D. Bremner, S. Ramaswami, and G. T. Wilfong. Computing constrained minimum-width annuli of point sets. *Computer-Aided Design*, 30(4):267–275, 1998.

[4] O. Devillers and P. A. Ramos. Computing roundness is easy if the set is almost round. *Int. J. Comput. Geom. Appl.*, 12:229–248, 2002.

[5] H. Ebara, N. Fukuyama, H. Nakano, and Y. Nakanishi. Roundness algorithms using the Voronoi diagrams. In *Proc. 16th The Canadian Conf. on Comput. Geom.*, volume 41, 1989.

[6] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Disc. Comput. Geom.*, 1:25–44, 1986.

[7] A. Eigenwillig and M. Kerber. Exact and efficient 2D-arrangements of arbitrary algebraic curves. In *Proc. 19th Annu. ACM-SIAM Symp. Disc. Alg.*, pages 122–131. Soc. for Industrial and Applied Math., 2008.

[8] J. Garcia-Lopez, P. A. Ramos, and J. Snoeyink. Fitting a set of points by a circle. *Disc. Comput. Geom.*, 20(3):389–402, 1998.

[9] D. Halperin, O. Setter, and M. Sharir. Constructing two-dimensional voronoi diagrams via divide-and-conquer of envelopes in space. ACS technical report ACS-TR-361601-01, TAU, 2008.

[10] V.-B. Le and D.-T. Lee. Out-of-roundness problem revisited. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):217–223, 1991.

[11] M. T. Marsh and D. A. Schilling. Equity measurement in facility location analysis: A review and framework. *European Journal of Operational Research*, 74(1):1–17, April 1994.

[12] K. Mehlhorn, S. Meiser, and R. Rasch. Furthest site abstract Voronoi diagrams. *Int. J. of Comput. Geom. Appl.*, 11(6):583–616, 2001.

[13] M. Meyerovitch. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional space. In *Proc. 14th Annu. Eur. Symp. Alg.*, pages 792–803, 2006.

[14] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley, NYC, 2nd edition, 2000.

[15] T. J. Rivlin. Approximating by circles. *Computing*, 21:93–104, 1979.

[16] U. Roy and X. Zhang. Establishment of a pair of concentric circles with the minimum radial separation for assessing roundness error. *Computer-Aided Design*, 24(3):161–168, 1992.

[17] K. Swanson, D.-T. Lee, and V. L. Wu. An optimal algorithm for roundness determination on convex polygons. In *Comput. Geom. Theory Appl.*, pages 601–609, 1993.

[18] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. *Comput. Geom. Theory Appl.*, 38(1–2):37–63, 2007. Special issue on CGAL.

[19] C. K. Yap. Exact computational geometry and tolerancing metrology. Technical Report SOCS-94.50, McGill School of Computer Science, 1994.

Generic implementation of a modular gcd over Algebraic Extension Fields

Michael Hemmer [*] Dominik Hülse [†]

Abstract

We report on several generic implementations for univariate polynomial gcd computation over the integers and, in particular, over algebraic extensions. Our benchmarks show that the generic implementation compares favorably to well established libraries. Even for the integer case our implementation is competitive to the one provided by the NTL, which does not support algebraic extensions. Our software is part of the new Polynomial package of CGAL release 3.4.

1 Introduction

For exact computation with non-linear geometric objects, such as semi-algebraic curves and surfaces, it is evident that the computation of the gcd of polynomials is one of the fundamental tools. This is the case for polynomials defined over rational coefficients as well as over algebraic extensions [6, 2]. It is known that methods based on modular arithmetic are indispensable for an efficient implementation [7, 5]. To the best of our knowledge there was no *generic* open source code available that supports algebraic extensions. Our software is *generic* in the sense that it uses C++'s template techniques [1] such as traits classes, and function objects. In particular, our code is independent from the coefficient type, even though we only report on algebraic extensions of degree 2 here. The presented implementation is part of the new Polynomial package of CGAL[1] release 3.4.

The paper is structured as follows: Section 2 provides an overview of the investigated algorithms. Section 3 presents the comparison of the different implementations including a comparison with the NTL[2] for integer polynomials and a comparison with Singular[3] for algebraic extensions of degree 2. Section 4 concludes the paper.

2 The Modular Methods

We now recall the principal ideas of modular gcd algorithms and the most fundamental modular methods of interest. In particular, we refer to Brown [3], who gave a solution for polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$. This was extended by Langemyr and McCallum [9] to polynomials over algebraic extensions using results from [14] in order to bound appearing denominators. Encarnacion [4] proposed a variant which uses rational reconstruction by Wang [12] in order to deal with denominators. We also present a hybrid approach that combines the ideas of both algorithms. All implementations are output sensitive, that is, the number of primes used within the computation depends on the size of the coefficients of the computed gcd and not on bounds based on the input polynomials.

2.1 Fundamentals

We restrict the presentation to univariate polynomials with integer coefficients. For more details study [3, sec. 4.3]. The principal idea is to compute the gcd with respect to several primes and to recover the original gcd in $\mathbb{Z}[x]$ or $\mathbb{Z}(\alpha)[x]$ using the Chinese Remainder theorem, e.g. see Knuth [7]. This avoids the exponential growth of coefficients in intermediate steps, whereas the actual gcd in practice has moderate coefficient size. Of course, it is important that the modular methods are output sensitive and do not rely on worst case bounds for the coefficient size in the final gcd, which is exponential [3].

For a given prime $p \in \mathbb{Z}$, let $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ be the Galois field with p elements and $\phi_p : \mathbb{Z} \to \mathbb{F}_p$ the field homomorphism defined by $\phi_p : x \mapsto (x \mod p)$. The homomorphism from $\mathbb{Z}[x]$ to $\mathbb{F}_p[x]$ induced by ϕ_p will also be denoted by ϕ_p. The image of ϕ_p will also be denoted as the *modular image* of some entity.

Let F be some polynomial in $\mathbb{Z}[x]$, we will use the following notation: $deg(F)$ - the degree of F; $lc(F)$ - the leading coefficient of F; $cont(F)$ - the content of F, that is, the gcd of all coefficients; $pp(F) = F/cont(F) \in \mathbb{Z}[x]$ - the primitive part of F; $monic(F) = F/lc(F) \in \mathbb{Q}[x]$ - the monic associate to F; $disc(F)$ - the discriminant of F.

2.2 GCD over the Integers

In this section we outline Brown's algorithm for polynomials with integer coefficients. Given $F_1', F_2' \in \mathbb{Z}[x]$, the algorithm computes $G' = gcd(F_1', F_2') \in \mathbb{Z}[x]$.

The core part of the algorithm is a while loop (step 5-13) that computes the gcd with respect to several primes until it is possible to recover the gcd using the Chinese Remainder Theorem.

[*]MPII Saarbrücken, `hemmer@mpi-inf.mpg.de`
[†]JoGU Mainz, `dominik.huelse@gmx.de`
[1]http://www.cgal.org/
[2]http://www.shoup.net/ntl/
[3]http://www.singular.uni-kl.de/

Algorithm 1: (Brown's algorithm) Given the polynomials F_1', $F_2' \in \mathbb{Z}[x]$ with $deg(F_1), deg(F_2) \geq 1$. Compute $G' \in \mathbb{Z}[x]$ the greatest common divisor of F_1' and F_2'.

```
(1)  Set $c_1 = cont(F_1'),\ c_2 = cont(F_2'), c = gcd(c_1, c_2)$.
(2)  Set $F_1 = F_1'/c_1,\ F_2 = F_2'/c_2$.
(3)  Set $f_1 = lc(F_1),\ f_2 = lc(F_2),\ \overline{g} = gcd(f_1, f_2)$.
(4)  Set n = 0, e = min(deg($F_1$),deg($F_2$)).
(5)  Let p be a new odd prime not dividing $\overline{g}$
(6)  Set $\tilde{g} = \phi_p(\overline{g}),\ \tilde{F_1} = \phi_p(F_1),\ \tilde{F_2} = \phi_p(F_2)$.
(7)  Invoke the Euclidean algorithm to compute
     $\tilde{G} = \tilde{g} \cdot\ gcd(\tilde{F_1}, \tilde{F_2})$, over $\mathbb{F}_p[x]$.
(8)  If deg($\tilde{G}$) = 0: set G = 1 and goto (15).
     If deg($\tilde{G}$) > e: (p is an unlucky prime) goto (5).
     If deg($\tilde{G}$) < e: (the former primes were unlucky)
         Set $n = 0,\ e = deg(\tilde{G})$.
(9)  Set n = n+1.
(10) If $n = 1$: set $(q, G^\star) = (p, \tilde{G})$ and goto (5).
(11) Use Chinese Remainder to update $(q, G^\star)$:
     $(q, G^\star) := chinese\_remainder((q, G^\star), (p, \tilde{G}))$.
(12) If the coefficients of $G^\star$ have changed goto (5).
(13) If $G^\star \nmid \overline{g} \cdot F_1$ or $G^\star \nmid \overline{g} \cdot F_2$ goto (5).
(14) Set G = pp($G^\star$).
(15) Output G' := cG;
```

For some (unlucky) primes it happens that the gcd loses a non trivial factor, which implies that the prime divides $lc(F_1)$ and $lc(F_2)$. The algorithm discards such primes in step 5. For other (unlucky) primes it happens that the gcd in $\mathbb{F}_p[x]$ contains additional factors. Therefore, the algorithm keeps track of $deg(\tilde{G})$, that is, it incorporates only those primes for which $deg(\tilde{G})$ is minimal (step 8).

Algorithm 1 deviates from the one of Brown [3] in the sense that it is output sensitive. Instead of computing as many primes as needed to guarantee a correct recovery of the gcd, it checks whether the recovered polynomial $G^\star$ becomes stable (step 12). If this is the case, $G^\star$ is in all probability the desired polynomial, which is verified in step 13. An idea which can, for instance, be found in Langemyr and McCallum [9].

Note that the Chinese Remainder can only recover polynomials in $\mathbb{Z}[x]$, that is, the algorithm must ensure that $G^\star$ is the image of a polynomial in $\mathbb{Z}[x]$. Therefore, $\tilde{G}$ is multiplied by $\tilde{g} = \phi_p(gcd(lc(F_1), lc(F2)))$ (step 7). Otherwise, $\tilde{G}$ as well as $G^\star$ would represent $monic(G)$ which is (in general) a polynomial in $\mathbb{Q}[x]$.

2.3 GCD over Algebraic Extension Fields

Given an algebraic number α and two polynomials F_1, $F_2 \in \mathbb{Z}(\alpha)[x]$, the following algorithms compute the greatest common divisor $G \in \mathbb{Z}(\alpha)[x]$ of F_1 and F_2 up to some constant factor. Note that it makes no sense to care about constant factors since $\mathbb{Z}(\alpha)$ does not support a gcd [5]

In principal, all algorithms have the same layout as Algorithm 1. However, a first fundamental difference is that the gcd in step 7 is computed over $\mathbf{R}_p = \mathbb{F}_p[t]/M_p$, where $M_p \in \mathbb{F}_p[x]$ is the modular image of the minimal polynomial M of α. In general M_p is not irreducible, thus in general $\mathbf{R}_p$ is not a field. Therefore, the computation in step 7 can fail, namely in that case that it needs to invert a zero divisor. If this happens, p is also considered as an unlucky prime and discarded.

We continue with details about the algorithm by Langemyr and McCallum [9], Encarnacion [4], and our hybrid approach combining the advantages of both algorithms.

The Algorithm of Langemyr and McCallum: In contrast to Algorithm 1, the main point is that the algorithm must take additional steps in order to ensure that the polynomial which is supposed to be recovered by the Chinese Remainder contains no denominators. In a first step, the input polynomials are normalized which removes superfluous constant factors and ensures that the leading coefficients are in $\mathbb{Z}$. This allows the computation of $\tilde{g}$ as in Algorithm 1 (step 7). However, in the presence of algebraic extensions, the multiplication with $\tilde{g}$ may not be enough to remove all denominators [14]. Therefore, $\tilde{G}$ is also multiplied by a multiplicative bound for these remaining denominators, namely $D = disc(M)$, the discriminant of the minimal polynomial of α. This finally ensures that $\tilde{G}$ is the modular image of a polynomial in $\mathbb{Z}(\alpha)[x]$, which can be recovered by the Chinese Remainder. For more details we refer to [11, 14, 8].

The Algorithm of Encarnacion: The algorithm does not multiply $\tilde{G}$ by any constant at all and the Chinese Remainder indeed tries to recover $monic(G) \in \mathbb{Q}(\alpha)[x]$ which is not possible. Instead, $G^\star$ is a polynomial in $\mathbb{Z}(\alpha)[x]$, where each coefficient is just in the same residue class as the corresponding coefficient of $monic(G)$. Therefore, the algorithm has an additional step that applies Wang's rational reconstruction [12, 13] to each coefficient in order to obtain $monic(G) \in \mathbb{Q}[x]$. Once the polynomial obtained in this step is stable, the verification (step 13) is applied.

The hybrid approach: For Langemyr and McCallum the weak point is that $gcd(f_1, f_2)D$ can be a very loose upper bound for the denominator of the gcd which causes the use of additional superfluous primes. Encarnacion's algorithm tries to avoid this, but has the overhead due to the additional rational reconstruction step which is performed in each round. In our benchmarks, see also Section 3, we observed that $gcd(f_1, f_2)$ is a good denominator bound in practice. Indeed, within all our examples the additional factor D was needed only once.

Our hybrid approach incorporates these observations in the sense that it modifies the algorithm by Langemyr and McCallum by using $gcd(f_1, f_2)$ as the denominator bound. This has the effect that the algorithm saves $O(log(D))$ rounds in almost all cases. However, for the unlucky case that D is indeed needed the algorithm would not terminate. Therefore, it uses the rational reconstruction as a fall back. More precisely, it calls Wang's algorithm if the fiftieth part of the accumulated time spent within the Chinese Remainder exceeds the time spent in the last call of Wang's algorithm. Hence, in practice our hybrid is as output sensitive as Encarnacion's algorithm but, de facto, without the extra costs of the rational reconstruction.

3 Benchmarks

Since our code is implemented within CGAL, we follow the *generic programming* paradigm using C++ templates and programming concepts such as traits classes and iterators. We introduced several traits classes to provide the functionality needed by the algorithms, for instance, providing the denominator bound required by Langemyr-McCallum. This abstracts from the actual coefficient type in use.

For the benchmarks we generated various families of 50 pairs of polynomials with fixed degree. Each pair is composed of three factors, the gcd and the two cofactors. All polynomials are random in the sense that their diced scalar coefficients have the desired bitsize. Within each family we always varied only one parameter, for instance, the bitsize of coefficients, or the degree of the gcd.

The benchmarks were measured on a Pentium(R) M processor 1.7 GHz with 512 KB cache under Linux and the GNU C++ compiler v3.4.6 with optimizations (`-O3`) and disabled assertions (`-DNDEBUG`). The used number type was `CORE::BigInt`.

3.1 Polynomials over the Integers

First we study the impact of a modification of the gcd and cofactor bitsize. For this purpose we generated polynomial pairs of degree 25 with gcd degree 1 and increasing bitsize of gcd and cofactors.

Generally we can say that Brown's algorithm performs far better than the old, non-modular implementation, hence we don't compare these two approaches. For a better quantification of the results we measured the same polynomials with the Computer Algebra Systems NTL and SINGULAR. Figure 1 shows, that the NTL implementation performs about twice as well as our generic implementation of Brown. Note, that there is another curve that also covers the time to convert our polynomials (i.e. the coefficients) to NTL polynomials and back. This curve is included for comparison with Singular, which uses NTL as well,

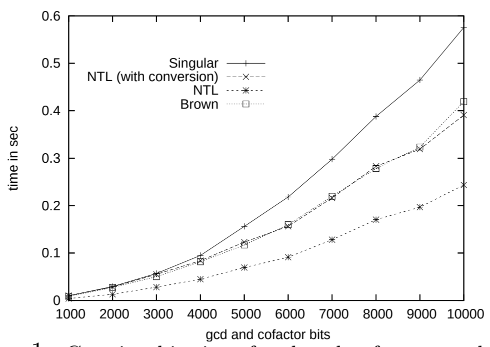


Figure 1: Growing bit size of gcd and cofactors, polynomial degree 25, gcd degree 1.

but who's conversion costs are apparently more expensive.

Furthermore, we generated polynomial pairs of degree 50 with 500 gcd bits and 5000 cofactor bits. The gcd degree ranges from 1 to 49. Figure 2 reveals a strange discontinuous behavior of the NTL: The first and the last 8 gcd degrees are computed faster than the others. Other test series show the same behavior. It seems that NTL uses two different approaches for the gcd computation. This has the consequence that our approach is even faster for moderate gcd degrees.

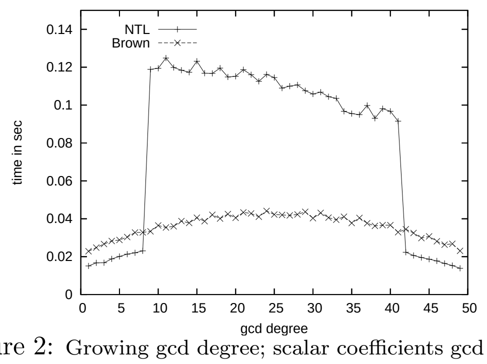


Figure 2: Growing gcd degree; scalar coefficients gcd 500 bit; cofactors 5000 bit; polynomial degree 50.

The behavior of Brown's algorithm can be explained as follows. Since degree and bit size of the polynomials are fixed, the time for the modular image is constant. With increasing degree of the gcd, the time spent in the Euclidean Algorithm decreases as it needs less steps whereas the Chinese Remainder has to recover more coefficients. These effects cancel out. The bow like form is due to the trial-division which is most expensive for moderate gcd degree.

3.2 Polynomials over Algebraic Extension Fields

To study the impact of a modification of the gcd bitsize we generated polynomial pairs of degree 10 with gcd degree 1 and 2000 cofactor bits. The Polynomials were defined over an algebraic extension of degree 2.

First of all, we can say that the hybrid approach performed far better than the non-modular implemen-

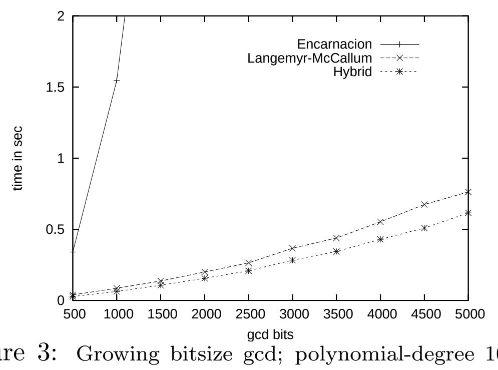


Figure 3: Growing bitsize gcd; polynomial-degree 10; gcd-degree 1; scalar coefficients cofactors 2000 bit.

tation. Hence, we don't consider the non-modular implementation.

Figure 3 shows that our hybrid approach performs better than the algorithms of Langemyr-McCallum (LM) and Encarnacion. Encarnacion's algorithm is not competitive, and for a sufficiently large bit size even slower than the old, non-modular implementation. This is due to the considerable runtime of Wang's rational reconstruction algorithm. For a higher bit size, the disadvantage of the LM approach due to the multiplication with the superfluous denominator bound becomes evident as well. The SINGULAR algorithm is the least successful, for a 500 bit gcd it needs already 128 sec. Hence, we refrained from including it in Figure 3.

A detailed decomposition of the total time spent in the hybrid algorithm is given in Figure 4. With growing gcd bits the algorithm needs more primes to reconstruct the coefficients. Additionally every call of the modular image, the Chinese remainder and the test division gets more expensive. The time for normalization and computing the denominator bound is slightly increasing, too.

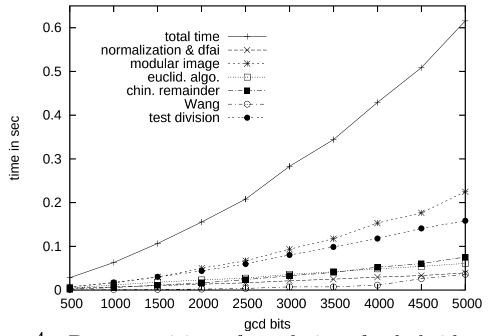


Figure 4: Decomposition of total time for hybrid approach: Growing bitsize gcd; polynomial-degree 10; gcd-degree 1; scalar coefficients cofactors 2000 bit.

4 Conclusions and Further Work

We have presented an open source implementation and comparison of several variants of gcd algorithms for algebraic extensions. Our benchmarks indicate that the hybrid approach has considerable advantages compared to the other implementations.

It is obvious that we should aim for a multivariate gcd in the spirit of [3]. As this is independent from the coefficient type, this should be straight forward.

We also expect some minor improvements for Encarnacion's algorithm, since the current implementation of Wang's algorithm does not take advantage of the known multiplicative denominator bound, as it is indicated in [10]. The algorithms are implement in CGAL, and part of the new Polynomial package of CGAL release 3.4.

References

[1] M. H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library.* Addison-Wesley, 1998.

[2] E. Berberich, M. Caroli, and N. Wolpert. Exact computation of arrangements of rotated conics. In *Proc. EWCG'07*, pages 231–234. Technische Universität Graz, 2007.

[3] W. S. Brown. On euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18(4):478–504, 1971.

[4] M. J. Encarnacin. Computing gcds of polynomials over algebraic number fields. *J. on Symbolic Computation*, 20(3):299–313, 1995.

[5] J. Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.

[6] M. Hemmer. *Exact Computation of the Adjacency Graph of an Arrangement of Quadrics.* Ph.D. thesis, Johannes Gutenberg-Universität Mainz, 2007.

[7] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, Reading, MA, 2nd edition, 1981.

[8] S. Landau. Factoring polynomials over algebraic number fields. *J. on Computing*, 14:184–195, 1985.

[9] L. Langemyr and S. McCallum. The computation of polynomial greatest common divisors over an algebraic number field. *J. on Symbolic Computation*, 8(5):429–448, 1989.

[10] M. Monagan. An almost optimal algorithm for rational reconstruction. In *Proc. ISSAC'04*, pages 243–249. ACM Press, 2004.

[11] P. S. Wang. Factoring multivariate polynomials over algebraic number fields. *Math. Comb.*, 30:1215–1231, 1978.

[12] P. S. Wang. A p-adic algorithm for univariate partial fractions. *Proc. SYMSAC '81*, pages 212–217, 1981.

[13] P. S. Wang, M. Guy, and J. Davenport. P-adic reconstruction of rational numbers. *SIGSAM Bulletin*, pages 2–3, 1982.

[14] P. J. Weinberger and L. P. Rothschild. Factoring polynomials over algebraic number fields. *J. Transactions on Mathematical Software*, 2(4):335–350, 1976.

Exact Delaunay graph of smooth convex pseudo-circles

Ioannis Z. Emiris* Elias P. Tsigaridas† George M. Tzoumas*

Abstract

We examine the problem of computing exactly the Delaunay graph of a set of possibly intersecting smooth convex pseudo-circles in the Euclidean plane given in parametric form. The diagram is constructed incrementally. We focus on InCircle, under the exact computation paradigm, and express it by a simple polynomial system, which allows for an efficient implementation by means of iterated resultants and a factorization lemma. Finally, we present examples with certain types of curves.

1 Introduction

Computing the Delaunay graph and its dual Voronoi diagram of a set of input sites in the plane has been studied extensively due to its numerous applications. However, few works have studied *exact* Voronoi diagrams[1] for curved objects. These can be critical in applications such as assembly, and surface reconstruction. In the case of circles, the exact and efficient implementation of [2] is now part of CGAL [1]. There is also VRONI, a very efficient and robust implementation, which relies on floating-point computations; a more recent implementation of which can treat (non intersecting) line segments and circular arcs [7].

Our own previous work [4] started with the study of non-intersecting ellipses, and proposed exact algebraic algorithms for all predicates required by the incremental algorithm of [8]. The resultant required had been implemented in MAPLE, and used a different polynomial system than the one in this paper. Moreover, some factorization properties had been observed without proof; this is settled below to yield an optimal method for resultant computation. In [5], the authors proposed a certified method for InCircle, relying on a Newton-like numerical subdivision, which exploits the geometry of the problem and exhibits quadratic convergence for non-intersecting ellipses.

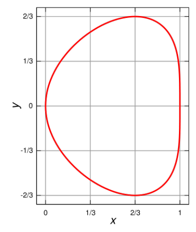


Fig. 1.1: The Bean curve $t \mapsto (\frac{1+t^2}{t^4+t^2+1}, \frac{t(1+t^2)}{t^4+t^2+1})$.

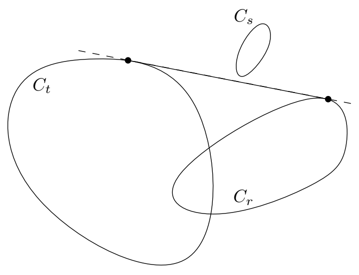


Fig. 1.2: Demonstration of DistanceFromBitangent.

In this paper, we extend previous results that considered only non-intersecting ellipses. We study the case of smooth convex, possibly intersecting, pseudo-circles. In this case, the bisector of two sites is a single curve homeomorphic to the open interval (0,1) [8, Thm.1]. We first model the definition of conflict from [8] as a circle inclusion test. Then, we examine the algebraic operations required for an efficient exact implementation of InCircle which shows high algebraic complexity dominating the overall algorithm. Proofs are omitted for lack of space, but can be found in [3].

Notation and preliminaries. Our input is smooth convex closed curves given in parametric form. An example of such a curve, the famous *bean* curve is in Fig 1.1. Smoothness allows the tangent (and normal) line at any point of the curve to be well-defined. We denote by C_t a smooth closed convex curve parametrized by t. We refer to a point p on C_t with parameter value $\hat{t}$ by $p_{\hat{t}}$, or simply by $\hat{t}$ when it is clear from context. By $C_t{}^\circ$ we denote the interior of curve C_t. $\mathbf{C_t}$ is a smooth convex object (site), so that if p denotes a point in the plane, $p \in \mathbf{C_t} \iff p \in C_t \cup C_t{}^\circ$. When we say that two sites intersect, we assume that their boundaries have at most two intersections, i.e. they form *pseudo-circles.* A curve C_t is given by the map

$$C_t : [a, b] \ni t \mapsto (X_t(t), Y_t(t)) = \left(\frac{F_t(t)}{H_t(t)}, \frac{G_t(t)}{H_t(t)}\right), \quad (1)$$

where F_t, G_t and H_t are polynomials in $\mathbb{Z}[t]$, with degrees bounded by d, and $a, b \in \mathbb{Q} \cup \{\pm\infty\}$. All algorithms, predicates and the corresponding analysis are valid for any parametric curve, even when the polynomials have different degrees, including the

*National Kapodistrian University of Athens, Hellas. {emiris, geotz} (AT) di.uoa.gr. Both authors are partially supported by Marie-Curie Network "SAGA", FP7 contract PITN-GA-2008-214584. G. Tzoumas is partially supported by State Scholarship Foundation of Greece, Grant No. 4631.

†INRIA Méditerranée, Sophia-Antipolis, France, elias.tsigaridas (AT) sophia.inria.fr. Partially supported by contract ANR-06-BLAN-0074 "Decotes".

[1]The Voronoi diagram cannot be represented "exactly", since it involves algebraic numbers. On the other hand, the dual Delaunay graph is represented exactly.

case of different denominators. We assume equations (1) for simplicity. Moreover, we assume that $H_t(t) \neq 0$, $t \in [a,b]$. For simplicity we write F_t instead of $F_t(t)$ and denote its derivative with respect to t as F'_t. When $d = 2$ the curves defined are conics: ellipses and circles are the only closed convex curves represented.

Basic predicates. Inserting a new site in the Voronoi diagram consists of the following: (i) Find a conflict between an edge of the current diagram and the new site, or detect that the latter is internal (hidden) in another site, in which case it does not affect the diagram. (ii) Find the entire *conflict region* (the part of the Voronoi diagram that changes due to the insertion of the new site) and update the dual Delaunay graph. It should be noted here that our main task is to compute an exact Delaunay graph of the input sites (which maintains exact topology information). Having computed an exact Delaunay graph, allows us to approximate the edges (bisectors) and vertices of the Voronoi diagram (the coordinates of which are algebraic numbers) within an arbitrary precision in order to be drawn on the screen. The exact graph allows us to mark correctly the degenerate cases, i.e., Voronoi vertices of degree > 3.

The above steps require the following predicates. (a) SideOfBisector: given two sites $\mathbf{C_t}$ and $\mathbf{C_r}$ and point q, determine the site closest to the point, under the Euclidean metric, (b) DistanceFromBitangent: given two sites, $\mathbf{C}_t$ and $\mathbf{C}_r$, decide the position of a third site, $\mathbf{C}_s$, with respect to the external bitangent line of the first two, that leaves both sites on the right, as we move from the tangency point of $\mathbf{C_t}$ to the tangency point of $\mathbf{C}_r$. The result of DistanceFromBitangent is either 1 if $\mathbf{C}_s$ lies on the same hyperplane as the sites $\mathbf{C}_r, \mathbf{C}_r$ w.r.t. their external bitangent, 0 if it is tangent to that line, but on the same hyperplane, and -1 otherwise (cf. fig. 1.2), (c) InCircle, and (d) EdgeConflictType. Some operations require two additional primitives: (i) computing a rational point inside the convex site and (ii) determining the position between two sites, i.e., whether they are separated. A detailed presentation of SideOfBisector, DistanceFromBitangent, EdgeConflictType and the primitives can be found in [3]. Predicate InCircle is presented in the next section.

Normal line. A point on the curve is denoted by $p_t = (X_t, Y_t)$. The equation of the line that supports the *normal* at p_t is $(N_t) : (x - X_t)X'_t + (y - Y_t)Y'_t = 0$. After substitutions and elimination of the denominators, we derive a polynomial $N_t(x,y,t) \in \mathbb{Z}[x,y,t]$; which is linear in x and y, of degree $\leq 3d-2$ in t.

2 InCircle

We first formalize the notion of conflict, caused by the addition of a new site [8], (see also fig. 2.1 and 2.2):

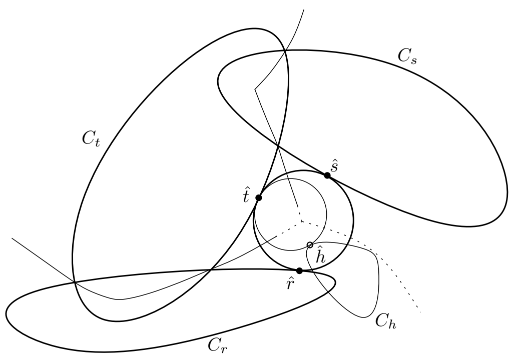


Fig. 2.1: Lem. 2: conflict of query site $\mathbf{C_h}$ with external Voronoi disk.

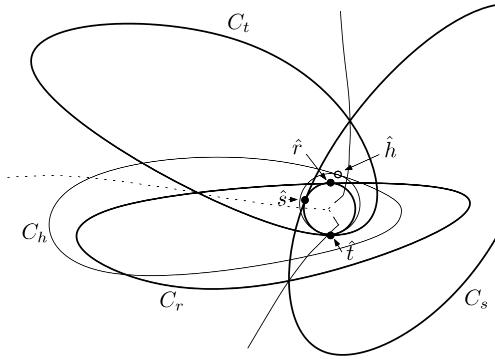


Fig. 2.2: Lem. 3: conflict of $\mathbf{C_h}$ with internal Voronoi disk; the Voronoi edges are solid, the conflict region (edge) is dotted.

Definition 1 *Given sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let V_{trs} be their Voronoi disk and $\mathbf{C_h}$ be a query site. If V_{trs} is an external Voronoi disk, then $\mathbf{C_h}$ is in conflict with V_{trs}, iff V_{trs} is intersecting ${C_h}^\circ$. If V_{trs} is an internal Voronoi disk, then $\mathbf{C_h}$ is in conflict with V_{trs}, iff V_{trs} is included ${C_h}^\circ$.*

Since the Voronoi circle in expressed algebraically, we cannot easily decide its relative position w.r.t. the query site i.e., by counting bitangent lines. Therefore, we model the above disk-site inclusion test as a circle-circle inclusion test.

Lemma 2 *Given convex sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let V_{trs} be an external Voronoi disk of theirs and $\hat{t}$ its tangency point on C_t. Let $\mathbf{C_h}$ be a query site and B_{th} an external bitangent disk of $\mathbf{C_t}$ and $\mathbf{C_h}$, tangent at $\hat{t}$ (and $\hat{h}$ resp.). Then $\mathbf{C_h}$ is in conflict with V_{trs} if and only if B_{th} is strictly contained in V_{trs}.*

It remains to compute the external bitangent circle B_{th}. There exist up to 6 bitangent circles, tangent at a given $\hat{t}$, for the case of ellipses, and up to a constant number for arbitrary convex sites. In [4], simple geometric tests are given to isolate the external bitangent circle among all bitangent circles to non-intersecting ellipses. Now, consider two intersecting sites $\mathbf{C_t}$, $\mathbf{C_r}$ and an external bitangent circle tangent at $\hat{t}$ of C_t, then: (i) $\hat{t} \notin C_t \cap C_r$ since it is the tangency point of an externally tangent circle, (ii) $\hat{t}$ lies on the arc of C_t bounded by the convex hull of $\mathbf{C_t}$ and $\mathbf{C_r}$ (iii) the tangent line of C_t at $\hat{t}$ intersects C_r.

These conditions were used to compute an external bitangent circle to non-intersecting ellipses [4], when the tangent line at the given tangency point intersects the other ellipse; this is always the case when the two sites intersect. Therefore, the geometric tests of [4] can be applied to pseudo-circles. These tests yield an arc on C_r, which contains the tangency point of the externally tangent circle.

Intersecting sites may also admit an internally tritangent Voronoi circle. In this case InCircle can be answered by the following lemma:

Lemma 3 *Given sites $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$, let V_{trs} be their internal Voronoi disk, and $\hat{t}$ its tangency point on C_t. Let $\mathbf{C_h}$ be a query site and B_{th} an internal bitangent disk of $\mathbf{C_t}$ and $\mathbf{C_h}$, tangent at $\hat{t}$ (and $\hat{h}$ resp.). Then $\mathbf{C_h}$ is in conflict with V_{trs} if and only if V_{trs} is strictly contained in B_{th}.*

Note that if $\hat{t} \notin \mathbf{C_h}$, then $\mathbf{C_h}$ is not in conflict with V_{trs} which can be used as an additional test in the implementation to quickly answer some cases.

Let us now identify an internal bitangent circle (among all bitangent circles). First, we observe that if an internal bitangent circle at point $\hat{t}$ of C_t exists, then $\hat{t} \in \mathbf{C_t} \cap \mathbf{C_r}$ and $\hat{r} \in \mathbf{C_t} \cap \mathbf{C_r}$ (the converse may not necessarily be true).

Lemma 4 *Given intersecting sites $\mathbf{C_t}$ and $\mathbf{C_r}$, consider their bitangent circle B_{tr} at points $\hat{t}$ and $\hat{r}$ respectively with $\hat{t}, \hat{r} \in \mathbf{C_t} \cap \mathbf{C_r}$. Then B_{tr} is an internal bitangent circle if and only if B_{tr} has the smallest radius among all bitangent circles of C_t and C_r tangent at $\hat{t}$, and the radius of B_{tr} is bounded by the radius of curvature of C_t at $\hat{t}$ and the radius of the self-bitangent circle of C_t at $\hat{t}$.*

In an implementation, the curvature constraint can be forced by comparing with the evolute point [6]. The self-bitangent circles can be computed by considering the bisector of a curve and itself, then by computing the (self-)bitangent circles and finally choosing the one with the smallest radius (in case of identical self-bitangent circle we can consider the evolute point).

Expressing the Voronoi circle. First, we consider the question of choosing, among all solutions of the polynomial system, the one corresponding to the Voronoi circle. The polynomial system expressing all circles tangent to $\mathbf{C_t}$, $\mathbf{C_r}$, $\mathbf{C_s}$ is:

$$\begin{aligned} N_t(x,y,t) = N_r(x,y,r) = N_s(x,y,s) &= 0 \\ M_{tr}(x,y,t,r) = M_{ts}(x,y,t,s) &= 0. \end{aligned} \quad (2)$$

The first 3 equations correspond to normals at points t, r, s on the 3 given sites. All normals go through the Voronoi vertex (x, y). The last two equations force (x, y) to be equidistant from the sites: each corresponds to the bisector of the segment between two footpoints. This system was also used in [9]. Notice that elimination of x, y from M_{tr}, N_t, N_r yields the bisector of two sites with respect to t, r.

A Voronoi circle is either externally or internally tritangent and its tangency points on C_t, C_r, C_s respectively have a CW or CCW orientation. Given a CCW orientation of the sites, either (t, r, s) or (t, s, r), we wish to identify the solution of the system that corresponds to that circle. First, we check if such a Voronoi circle exists. This is a generalization of the Existence sub-predicate of [2] to pseudo-circles. It can be performed for an external Voronoi circle without solving system (2) as follows: Sites C_t, C_r, C_s admit an external CCW Voronoi circle, iff there are at least two negative results of DistanceFromBitangent evaluated at triplets (C_t, C_r, C_s), (C_r, C_s, C_t) and (C_s, C_t, C_r) in this cyclic order (proof by enumeration). Checking that C_t, C_r, C_s admit an internal CCW Voronoi circle is more complex: First, we compute their intersection, which must be nonempty. Then, their intersection must have a CCW sequence of arcs on its boundary. Finally, we have to verify that system (2) has a solution corresponding to the Voronoi circle.

Solving system (2) over the reals, yields a set of solution vectors in $\mathbb{R}^5$. Only one solution vector contains the Voronoi vertex and the corresponding tangency points. There exist solution vectors with CW orientation, but also with CCW orientation which do not correspond to the Voronoi circle we are looking for, but to some other tritangent circle. At this point, we already know that an external Voronoi circle (with CCW orientation) exists, or that an internal Voronoi circle might exist. To eliminate irrelevant solutions, consider the tangency points $p_{\hat{t}}$, $p_{\hat{r}}$, $p_{\hat{s}}$ for a solution triplet $\hat{t}, \hat{r}, \hat{s}$. The tangency points corresponding to the Voronoi circle satisfy CCW($p_{\hat{t}}$, $p_{\hat{r}}$, $p_{\hat{s}}$).

Now we distinguish an external and an internal tritangent circle from the rest of the tritangent circles. The tangency points define the former iff the tangent line of the Voronoi circle at each tangency point separates its adjacent site from the other two tangency points. Even if the tangent line intersects the other sites, the tangency points are still separated. Checking that the tangency points correspond to an internal circle can be performed by applying lemma 4. Finally, before proceeding with algebraic analysis, it has to be noted that in an implementation, the certified algorithm of [5] can be adapted in order to speed up the operations.

We examine system (2) targeting an efficient algorithm for its solution. In general, the *resultant* of $n+1$

polynomials in n variables is an irreducible[2] polynomial in the coefficients of the polynomials which vanishes precisely when the system has a complex solution. Since it is impossible to compute the resultant of 5 general polynomials as a determinant, we compute it by successive Sylvester determinants, which are optimal formulae for the resultant when $n = 1$. This method typically produces extraneous factors but, by exploiting the fact that some of the polynomials are linear, and that none contains all variables, we shall be able to predict all such factors. The following theorem, whose proof is in [3], provides the factorization of the resultant, in the case of conics.

Theorem 5 *We assume the resultant of (2) is nonzero and denote it by* Π. *Then,* $\mathsf{Res}_{xy}(R_1, R_2, N_t) = \Pi(t) H_t^{40} (G_t H_t' - G_t' H_t)^{36}$, *where,* $R_1 = \mathsf{Res}_r(M_{tr}, N_r)$, $R_2 = \mathsf{Res}_s(M_{ts}, N_s)$, *and* Π *is of degree 184.*

Corollary 6 *We are given* $R_0, R_1, R_2 \in \mathbb{K}[x, y]$, *where the total degree of* R_1 *and* R_2 *is* n *in* x, *in* y, *and in* x *and* y *together, and* $R_0 = Dy + Ax + C$, *where* $AD \neq 0$, *then* $\mathsf{Res}_x(\mathsf{Res}_y(R_0, R_1), \mathsf{Res}_y(R_0, R_2)) = D^{n^2} \mathsf{Res}_{xy}(R_0, R_1, R_2)$.

It follows that the degree of the resultant of (2) for general parametric curves, as in (1), is bounded by $(3d-2)(5d-2)(9d-2)$, after dividing out the factor of $(H_t(G_t H_t' - G_t' H_t))^{(5d-2)^2}$. A more careful analysis may exploit cancellations to yield a tighter bound.

3 Conclusion

We conclude this paper by applying the proposed algorithms for the resultant on various types of curves. Results are summarized in table 1. The first column shows the type of curve, the second its degree, the third the time in sec, the fourth the degree of the resultant and the last column shows the (non-tight) bound of our general formula. All experiments were run on a P4 2.4GHz.

First, we took the *bean* curve of fig. 1.1 and applied simple affine transformations yielding very small (5-bit) coefficients. Then, we computed the resultant of such triplets. The long runtimes indicate that working with high-degree curves might be non-practical. We additionally considered the case of three random *conics* with small (10-bit) coefficients. Note that in this case ($d = 2$) we have a tight bound of 184, while the general formula yields 512.

As a future work we may extend this approach to working with piecewise smooth polynomial curves. The critical part is to determine the "piece" of the function where one applies the algebraic predicate. This may be achieved by a numeric technique, as the one in [5], which determines the involved pieces in the case of InCircle. We need similar methods for the other predicates too. The tangency points of the Voronoi circle lie on such polynomial pieces and the resultant formulation is simpler because there are no denominators. We have applied the resultant computation on polynomial branches of degree two and three (*B-splines*), with small 5-bit coefficients. The runtime is less than 1 sec for $d = 2$ and less than 10 sec when $d = 3$. Therefore, an efficient exact implementation is still possible and we can benefit from the increased flexibility that piecewise functions offer.

Curves	d	**time**	**resultant** d	**bound**
Beans	4	570.0	2632	6120
Conics	2	8.5	184	512
B-splines	3	9.9	404	2275
B-splines	2	0.7	93	512

Table 1: Examples with various curves

[2]Irreducibility occurs for generic coefficients. Below, certain resultants are factorized, because the given polynomials do not have generic coefficients.

References

[1] CGAL: Computational geometry algorithms library. http://www.cgal.org.

[2] I.Z. Emiris and M.I. Karavelas. The predicates of the Apollonius diagram: algorithmic analysis and implementation. *Comp. Geom.: Theory & Appl.*, 33(1-2):18–57, 2006. Spec. Issue robust geom. algorithms & implementations.

[3] I.Z. Emiris, E.P. Tsigaridas, and G.M. Tzoumas. Exact Delaunay graph of smooth convex pseudo-circles: General predicates, and implementation for ellipses. Available from http://www.di.uoa.gr/∼geotz/, 2008.

[4] I.Z. Emiris, E.P. Tsigaridas, and G.M. Tzoumas. Predicates for the Exact Voronoi Diagram of Ellipses under the Euclidean Metric. *Int. J. Comp. Geom. & Apps*, 18(6):567–597, 2008. Spec. Issue SoCG'06.

[5] I.Z. Emiris and G.M. Tzoumas. Exact and efficient evaluation of the InCircle predicate for parametric ellipses and smooth convex objects. *Comput. Aided Des.*, 40(6):691–700, 2008.

[6] I. Hanniel, R. Muthuganapathy, G. Elber, and M.-S. Kim. Precise Voronoi cell extraction of free-form rational planar closed curves. In *Proc. 2005 ACM Symp. Solid and phys. modeling*, pages 51–59, 2005.

[7] M. Held and S. Huber. Topology-Oriented Incremental Computation of Voronoi Diagrams of Circular Arcs and Straight Line-Segments. *Computer Aided Design*, (to appear), 2008. Spec. Issue on Voronoi diagrams.

[8] M. I. Karavelas and M. Yvinec. Voronoi diagram of convex objects in the plane. In *Proc. Europ. Symp. Algorithms*, LNCS, pages 337–348. Springer, 2003.

[9] R. Ramamurthy and R. Farouki. Voronoi diagram and medial axis algorithm for planar domains with curved boundaries - II: detailed algorithm description. *J. Comput. Appl. Math.*, 102(2):253–277, 1999.

On the Optimality of Spiral Search

Elmar Langetepe*

Abstract

Searching for a point in the plane is a well-known search game problem introduced in the early eigthies. The best known search strategy is given by a spiral and achieves a *competitive ratio* of 17.289... It was shown by Gal [7] that this strategy is the best strategy among all *monotone* and *periodic* strategies. Since then it was unknown whether the given strategy is optimal in general. This paper settles this old open problem and shows that spiral search is indeed optimal.

Keywords: Search games, motion planning, spiral search, competitive analysis, lower bound

1 Introduction

Search games (i. e., games where two players, a searcher and a hider, compete with each other) are studied in many variations in the last 60 years since the first work by Koopman in 1946. For example, Bellman [3] introduced the search for an immobile hider located on the real line with a known probability distribution, Gal [7] and independently Baeza-Yates et al. [2] solve this problem for a uniformly distributed location of the hider. The book by Gal [7] and the reissue by Alpern and Gal [1] gives a comprehensive overview on results on search games.

For analysing the efficiency of a search startegy we use the competitive framework which was introduced by Sleator and Tarjan [10], and used in many settings since then, see for example the survey by Fiat and Woeginger [6] or, for the field of online robot motion planning, see the surveys [9, 8].

We consider a special search game problem introduced by Gal [7], namely *searching for a point in the plane*. Starting from a fixed origin O we move along a path Π through the plane. Let us assume that there is an unknown target point t and let p_t denote the first point on Π so that t lies on the line segment between O and p_t. We detect t at point p_t. This means, that we *sweep* the plane until finally the unknown target t is found, see Figure 1. We assume that the target point is at least one step away from the start.

The efficiency of the search path Π is given by the worst-case target, $C := \sup_t \frac{|\Pi_O^{p_t}|}{|Ot|}$, the constant C is

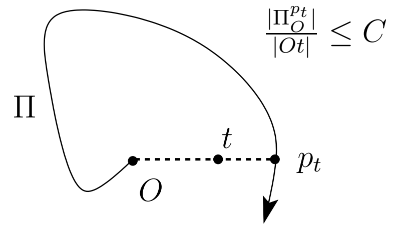


Figure 1: Searching for a point in the plane.

called the *competitive ratio* of Π. It was shown by Gal [7] that a spiral strategy is the best strategy among all *monotone* and *periodic* strategies. A strategy S represented by its radius vector $X(\theta)$ is called periodic and monotone, if θ is always increasing and X also satisfies $X(\theta+2\pi) \geq X(\theta)$. Gal states that it might be a *complicated task* to show that there is a periodic and monotone optimal strategy, a lower bound remains open.

2 Spiral search

We consider a logarithmic spiral, Π, which is given in polar coordinates by $(\varphi, \cdot e^{\varphi \cot(\alpha)})$ for $-\infty < \varphi < \infty$, see Figure 2 for an example with $\alpha \leq \pi/2$. The angle $\alpha \leq \pi/2$ expresses the excentricity of the spiral. The length of the spiral from the center O to some point q is given by $\frac{1}{\cos\alpha}|Oq|$, for details see [4].

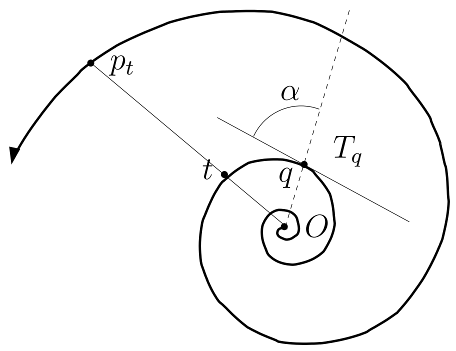


Figure 2: A logarithmic spiral and the worst-case situation.

The spiral expands successively and for every target point t there will be a first point p_t on the spiral so that the segment p_tO will hit the target t for the first time. Obviously, the worst case for the competitive ratio is given, if we miss the target t arbitrary close to

*University of Bonn, Institute of Computer Science I, 53117 Bonn, Germany

$(\varphi, e^{\varphi \cot \alpha})$ and detect t at $p_t = (\varphi + 2\pi, e^{(\varphi+2\pi)\cot\alpha})$, see Figure 2. Altogether, the worst-case ratio for the spiral is given by

$$\frac{|\Pi_O^{p_t}|}{|Ot|} = \frac{1}{\cos\alpha} e^{2\pi\cot\alpha}\,.$$

Optimization on α gives a competitive ratio of $17.289\ldots$ for $\cot\alpha = 0.15540\ldots$ This strategy is the best strategy among all *periodic and monotone* strategies, see Alpern and Gal [1].

Theorem 1 *The optimal spiral for searching a point in the plane achieves a competitive ratio of* $17.289\ldots$

3 Lower bound construction

The maximal lower bound known so far is $17.079\ldots$ and was presented in [5] in the context of searching for rays in the plane. The lower bound construction here consists of the following steps.

1. We consider a discrete version of the problem using a bundle of m rays that emanate from the origin, this was also used in [5].
2. We simplify the problem for m rays in some steps which results in a ratio not greater than the continous version of the problem.
3. We show that the optimal solution of the simplified problem has to be monotone and periodic for every m.
4. For every m we compute the optimal strategy using the framework of Gal. The optimal competitive ratio goes to 17.289... if m goes to infinity.

Let us first consider a discrete version of the problem using a bundle of m rays that emanate from the origin and which are separated by an angle $\alpha = \frac{2\pi}{m}$, see Figure 3. The target will be on one of the rays. Again, the goal is detected, if it is swept by the radius vector of the trajectory, i.e., t is hit by a segment p_tO and p_t is visited on the corresponding ray. Note that if m goes to infinity we are back to the continous version of the problem. But we can neither assume that we have to visit the rays in a periodic order nor that the depth of the visits increases in every step.

We represent a search strategy, S, as follows: In the ith step, the searcher hits a ray—say ray l—at distance x_i from the origin, moves a distance $\beta_i x_i - x_i$ along the ray l, and leaves the ray at distance $\beta_i x_i$ with $\beta_i \geq 1$. Then, it moves to the next ray within distance $\sqrt{(\beta_i x_i)^2 - 2\beta_i x_i x_{i+1}\cos\gamma_{i,i+1} + x_{i+1}^2}$, see Figure 3. Note that any search strategy for our problem can be described in this way. Let us assume that the ray l was visited up to distance $\beta_k x_k$ and is visited the next time at index J_k. The worst case occurs if the searcher slightly misses the goal while visiting ray l up to distance $\beta_k x_k$. Instead, it finds the goal at

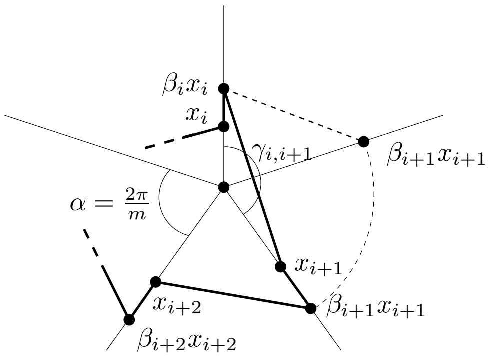


Figure 3: A bundle of rays, a reasonable strategy and a shortcut.

step x_{J_k} on ray l arbitrarily close to $\beta_k x_k$. Either we have $x_{J_k} > \beta_k x_k$; that is, the searcher discovers the goal in distance x_{J_k} on ray l, or we have $x_{J_k} < \beta_k x_k$. In the latter case, the searcher moves $\beta_k x_k - x_{J_k}$ from x_{J_k} and finds the goal by accident. Altogether, the competitive ratio, $C(S)$, is greater than

$$\frac{\sum_{i=1}^{J_k-1} \beta_i x_i - x_i + \sqrt{(\beta_i x_i)^2 - 2\beta_i x_i x_{i+1}\cos\gamma_{i,i+1} + x_{i+1}^2}}{\beta_k x_k}\,. \tag{1}$$

We simplify the problem for m rays in some steps. We do not change the movement of the strategy but we will improve the ratio. Instead of the distance

$$\sqrt{(\beta_i x_i)^2 - 2\beta_i x_i x_{i+1}\cos\gamma_{i,i+1} + x_{i+1}^2} + \beta_{i+1}x_{i+1} - x_{i+1}$$

from $\beta_i x_i$ to x_{i+1} and then to $\beta_{i+1}x_{i+1}$ between two arbitrary successive rays we let this distance *shrink* to $\sqrt{(\beta_i x_i)^2 - 2\beta_i x_i \beta_{i+1}x_{i+1}\cos\frac{2\pi}{m} + (\beta_{i+1}x_{i+1})^2}$. This would be the distance between two neighboring rays without slipping along the second ray, see the dashed line in Figure 3. The new distance is obviously not greater than the original one, of course it might be the same for $\beta_{i+1} = 1$ and two neighboring rays, which means $\gamma_{i,i+1} = \frac{2\pi}{m}$. We change only the path length for the ratio but we do not change the movements of the given strategy. Therefore, the ratio (1) cannot increase, because the numerator will not increase.

There is only one problem in this reformulation concerning the last value of the sum in the numerator of ratio (1). The last step of the strategy (before detecting the goal at $\beta_k x_k$) goes from $\beta_{J_k-1}x_{J_k-1}$ to x_{J_k} and not directly to $\beta_{J_k}x_{J_k}$ and this step might indeed be smaller than the distance from $\beta_{J_k-1}x_{J_k-1}$ to $\beta_{J_k}x_{J_k}$. Therefore, we allow—only for the computation of the ratio—that the last value in the numerator of (1) is given by the distance of the shortest path from $\beta_{J_k-1}x_{J_k-1}$ to a neighboring ray, which is $\beta_{J_k-1}x_{J_k-1}\sin\frac{2\pi}{m}$. One can imagine that this last step will have no influence if we let m go to infinity at the end. Again, we do not change the movement of the strategy, we only improve the ratio.

For convenience, from now on we denote $\beta_i x_i$ by y_i. Altogether, we would like to minimize

$$\frac{y_{J_k-1}\sin\frac{2\pi}{m}+\sum_{i=1}^{J_k-2}\sqrt{y_i^2-2y_iy_{i+1}\cos\frac{2\pi}{m}+y_{i+1}^2}}{y_k}. \tag{2}$$

Obviously, the ratio (2) is never greater than ratio (1), only the numerator might be smaller.

The simplification above results in the following much more simple interpretation of the problem. We can assume that there are only two rays with angle $\frac{2\pi}{m}$ between them and the agent moves successively from one to the other, see an example (bold line) in Figure 4 for $m = 5$ up to 11 steps. For every visit the agent only decides which ray, l, of the original m rays it would visit in the m-ray setting. Note that this also means that the important indices J_k are fixed in that way. Remember that the ray visited with depth x_k is still visited the next time at index J_k.

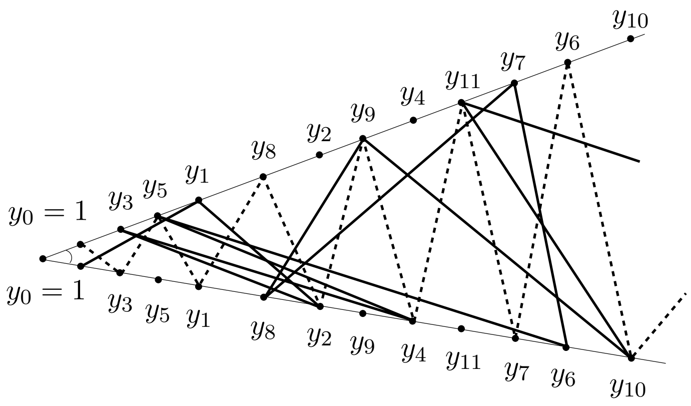


Figure 4: The problem for $m = 5$ rays interpreted by successive visits on two rays. The Strategy S (bold line) can be replaced by the sequence S' (dashed line).

Let us now assume that we have an optimal strategy $S = (y_1, y_2, y_3, y_4, \ldots)$ and given values J_k for this problem. The worst-case ratio of S is the worst-case ratio of (2). The target is at least one step away from the origin. The starting depth on every ray is 1. At every step y_i the rays are visited up to a certain depth. For the optimal sequence S we can now choose a *visiting order* (i.e., we improve the indices J_k) in such a way that at every step y_i the ray with the smallest current depth is visited next. Obviously, this will keep the ratio (2) as small as possible regardless how the original visiting order was. If a ray with smallest current depth is visited later, some more steps were made and the ratio (2) will increase. Note that we do not change the movements of the strategy.

For example, in Figure 4 in the optimal visiting order the first five steps $y_1, \ldots, y_5$ visits the five *imaginary* rays successively since the starting depth on every ray was 1. For distance y_6 we set $J_3 = 6$ since y_3 is the smallest current depth on all rays. This means that y_6 visits the same ray as y_3. Further on we obtain $J_5 = 7$ since y_5 is the smallest current depth at step y_7, then we have $J_1 = 8$, $J_8 = 9$, $J_2 = 10$, $J_9 = 11$ and so on. This is the best visiting order for S.

If two rays have exactly the same current depth, we choose the one which was visited earlier. Note that we do not change the movements of the strategy and it might still happen that y_{i+1} is smaller than y_i. But we know that at step y_{i+1} there is at least one ray that has current depth smaller than y_{i+1}, otherwise with distance y_{i+1} we would not detect more goals and step y_{i+1} was needless, we can skip such a step. Of course, after a step with $y_{i+1} < y_i$ the value y_{i+1} might be the smallest current depth on all rays. Then its ray should be visited again in the next step y_{i+2}. We simply allow to visit the same ray in two successive steps which is not possible in the original m-ray version. This additional freedom can only improve the ratio (2).

Since J_k is the index where the ray of y_k is visited next, for $J_l < J_j$ we now have $y_l \leq y_j$. The next idea is that we really visit the two rays in an *increasing* order. In the example of Figure 4 the strategy $S = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11} \ldots)$ gives $J_3 = 6$, $J_5 = 7$, $J_1 = 8$, $J_8 = 9$, $J_2 = 10$, $J_9 = 11$ and so on. Note that the first $5 = m$ elements of S visit the 5 rays successively. Thus, the sequence S now will be reordered to $S' = (y'_1, y'_2, y'_3, y'_4 \ldots)$ with $y'_1 = y_3$, $y'_2 = y_5$, $y'_3 = y_1$, $y'_4 = y_8$, $y'_5 = y_2$ $y'_6 = y_9$, $y'_7 = y_4$, $y'_8 = y_{11}$, $y'_9 = y_7$ and so on.

For S' we can again use a *visiting order* induced by the smallest current depth, this can not increase the ratio for S' as mentioned before. This means that the rays are now visited in successive order and with increasing distance. For index n in S', the index J'_n is exactly $n + m$. Thus, S' is monotone and periodic and the ratio is given by

$$\frac{y'_{n+m-1}\sin\frac{2\pi}{m}+\sum_{i=1}^{n+m-2}\sqrt{{y'_i}^2-2y'_iy_{i+1}\cos\frac{2\pi}{m}+{y'_{i+1}}^2}}{y'_n}. \tag{3}$$

We still have to show that S' is not worse than S with respect to the competitive ratio. The worst case for depth y_k on S is attained for the next visit at index J_k. In S' the distance y_k might be visited in an earlier step than in S, that is $y'_n = y_k$ and $n \leq k$. For index n in S', the index J'_n equals exactly $n + m$ as already seen. We would like to compare the ratios for y_k in S and $y'_n = y_k$ in S'. It is easy to see that for the sequence S also $J_k = n + m$ has to be fulfilled, since the visit order in S was induced by current smallest depth. For example, in the sequence S' in Figure 4, the ray of $y'_2 = y_5$ was visited at index 7 again and in the sequence S the ray of y_5 was also visited again at index 7. Thus, for comparing the ratios for $y'_2 = y_5$ we have to use the sums in the ratios of S and S' up to the same index 7 but with some different elements.

This holds in general. We consider the sequence S. Since in the beginning the m rays were visited successively by the depths $y_1, \ldots, y_m$ one of the rays is visited with depth y_{m+1} in the next step and y_{m+1} is one of the new current depths of the rays. Generally at every step y_k, the element y_k will be used on one of the rays and therefore one current depth is exchanged by y_k. In the sorted order of $y_1, \ldots, y_{k-1}$ the $k - m$ greatest element will be exchanged, which is element ${y'}_{k-m}$. Moreover, up to index k the sequences S and S' will have at most $m - 1$ different entries and these entries are greater in S than in S'.

Altogether, this means that the number of visits for the worst case at y_k or $y'_n = y_k$ is the same (namely $J_k = n + m = J'_n$) and only at most $m - 1$ elements in S are greater than that of S' up to index $n + m$.

We consider a simple shortest path problem. Let a sequence of elements $S = (y_0, y_1, y_2, y_3, \ldots, y_{n+m})$ be given. We use images of the corresponding points on both rays as already depicted in Figure 4. The task is, to compute a shortest path that starts at the smallest element y_j in S and visits exactly one of the two images for every element y_i but changes sucessively from one ray to the other.

We would like to show that the shortest path has to visit the rays in an increasing order, see the dashed path in Figure 4. This can be shown by induction on the length of S. For two elements in S this is trivial. So we consider a sequence S with $n + 1$ elements and assume that the statement holds for all sequences with less than n elements. By triangle inequality we can show that the shortest path always starts with the segment of smallest slope. Now we delete the smallest element y_j out of S and the corresponding shortest path visits the remaining elements in increasing order starting at the second smallest value y_i. The image of y_j is closer to y_i than to any other part of the shortest path for $S \setminus \{y_j\}$, therefore we can combine the segment $y_j y_i$ with the shortest path starting at y_i. We obtain an overall shortest path that visits the images in increasing order.

Unfortunately, S and S' might have $m-1$ different elements. All these elements in S are greater than the corresponding elements in S'. The corresponding shortest path problem for S will result in a path of smaller length, if we move a couple of points with greatest distance closer to the origin. Therefore, we substitute the differing elements of S by the corresponding elements of S'.

Altogether, we conclude that $y_{n+m-1} \sin \frac{2\pi}{m} + \sum_{i=1}^{n+m-2} \sqrt{y_i^2 - 2y_i y_{i+1} \cos \frac{2\pi}{m} + y_{i+1}^2}$ is greater than or equal to

$$y'_{n+m-1} \sin \frac{2\pi}{m} + \sum_{i=1}^{n+m-2} \sqrt{{y'_i}^2 - 2y'_i y_{i+1} \cos \frac{2\pi}{m} + {y'_{i+1}}^2}$$

and the sequence S' is optimal because for every $y'_n = y_k$ the ratio (3) is not greater than the ratio (2).

In principle, we are already done, because as m goes to infinity we get arbitrarily close to the *searching-for-a-point-in-the-plane* problem and there is always an optimal solution that is periodic and monotone. Thus, together with the previous result of Gal, spiral search is optimal.

But we can also proceed more directly as follows: We compute an optimal sequence S' for ratio (3). Fortunately, S' is periodic and monotone and fulfills some other nice properties (for example unimodality) so that a general framework of Gal is applicable, see [7, 1]. This means that (3) is minimized by an exponential sequence $y'_i = a^i$. Simple arithmetic shows that the ratio is given by $f(a, m) = a^{m-1} \sin \frac{2\pi}{m} + \frac{a^{m-1}}{a-1} \sqrt{1 - 2a \cos \frac{2\pi}{m} + a^2}$. Thus, we can analytically find the value $a_{\min}$ that minimizes $f(a, m)$. For increasing m the corresponding value $f(a_{\min}, m)$ converges to 17.289... For example, for $m = 5000$ we compute $a_{\min} = 1.000195303\ldots$ and $f(a_{\min}, 5000) = 17.289\ldots$

Theorem 2 *Spiral search is optimal.*

References

[1] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous.* Kluwer Academic Publications, 2003.

[2] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Inform. Comput.*, 106:234–252, 1993.

[3] R. Bellman. An optimal search problem. *SIAM Rev.*, 274(5), 1963.

[4] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik.* Verlag Harry Deutsch, Frankfurt am Main, 5th edition, 2000.

[5] A. Eubeler, R. Fleischer, T. Kamphans, R. Klein, E. Langetepe, and G. Trippen. Competitive online searching for a ray in the plane. In *Robot Navigation*, Dagstuhl Seminar Proceedings, 2006.

[6] A. Fiat and G. Woeginger, editors. *On-line Algorithms: The State of the Art*, volume 1442 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1998.

[7] S. Gal. *Search Games*, volume 149 of *Mathematics in Science and Engeneering.* Academic Press, New York, 1980.

[8] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. In *Sensor Based Intelligent Robots*, volume 2238 of *Lecture Notes Comput. Sci.*, pages 245–258, Berlin, 2002. Springer.

[9] N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar. Robot navigation in unknown terrains: introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

[10] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.

Computing Maximal Islands

C. Bautista-Santiago* J.M. Díaz-Báñez† D. Lara‡ P. Pérez-Lantero § J. Urrutia¶ I. Ventura‖

Abstract

Given a set S of red and blue points on the plane in general position, an island $I(S, C) \subseteq S$ is the intersection of S and a convex region C. We study the problem of finding a maximal island according to certain criterium. For instance, a largest monochromatic island $I(S, C)$ is a maximum cardinality subset of S, such that every point of $I(S, C)$ belongs to the same chromatic class. An $O(n^3)$-time and $O(n^2)$-space algorithm is proposed to find one such subset. Our approach can be adapted to find monochromatic islands which maximize parameters such as the area, perimeter.

1 Introduction

In this paper, S will always denote a set of n points on the plane in general position such that its elements are classified into two classes or *colors*, say *red* and *blue.* A subset $\mathcal{I}$ of S is called an island of S if there is a convex set C such that $\mathcal{I} = S \cap C$. We say that an island of S is a blue (resp. red) island of S if all of its elements are blue (resp. red).

In this paper we study the problem of finding islands of S that are optimal according to some parameters, e.g. find a blue island of S with maximum cardinality.

The problem of finding the largest monochromatic island of a point set S (*LMIP*) was studied in [17], where an $O(n^3 \log n)$ time and $O(n^2)$-space algorithm to solve this problem is given. We present here an $O(n^3)$ time and $O(n^2)$-space algorithm to solve the LMIP problem which also solves all the problems studied in [17]. Our algorithms can also handle the weighted version of our main problem in which the elements of S are assigned weights (usually integral values).

We observe that when the labels of the elements of S are chosen carefully, we can solve problems apparently unrelated. For instance, if we label all the blue points of S with 1, and the red points with -1, a maximum weight island is an island with maximum discrepancy (absolute difference between blue and red points) [11]. If we label the blue points of S with 1, and the red points of S with $-\infty$, the maximum weight island is the largest monochromatic island. Our algorithm can also be easily adapted to solve the following problems: Find the monochromatic island $\mathcal{I}$ of S maximizing the number of vertices on the convex hull of $\mathcal{I}$; find a monochromatic island $\mathcal{I}$ of S that maximizes the area, or the perimeter of the convex hull $Conv(\mathcal{I})$ of $\mathcal{I}$, discrepancy and others.

An outline of the paper is as follows. In Section 2 we present the algorithm in detail. A generalization for solving other optimization problems is stated in Section 3. Finally, in Section 4 we show an application of our algorithm to give an heuristic method for a red-blue set covering problem.

1.1 Related work

A key problem in computational geometry is the identification of subsets of a point set having particular properties. Erdös and Szekeres [16] asked whether there was a value $f(k)$ such that all sets of at least $f(k)$ points in general position in the plane determine a convex k-gon. They give upper and lower bounds for $f(k)$.

Several papers have also studied the algorithmic aspects of problems of this kind, e.g. see [4, 10]. Algorithms are also known for finding subsets of points with k elements that minimize parameters such as diameter, or perimeter of the convex hull of these subsets [2, 15].

Our motivation to study the LMIP problem arises from applications in data mining, statistical clustering, pattern recognition or data compression. In data mining and classification problems, a natural method for analyzing data is to select prototypes representing different data classes. A standard technique for achieving this is to perform cluster analysis on the training data [12]. The clusters can be obtained using simple geometric shapes. Aronov and Har-Peled [3] and Eckstein et al. [13] considered circles and axis-

*Instituto de Matemáticas, Universidad Nacional Autónoma de México, crevel@uxmcc2.iimas.unam.mx

†Departamento Matemática Aplicada II, Universidad de Sevilla, dbanez@us.es, partially supported by grant FEDER-MTM2006-03909.

‡Instituto de Matemáticas, Universidad Nacional Autónoma de México, dlara@uxmcc2.iimas.unam.mx

§Departamento de Computación, Universidad de La Habana, pablo@matcom.uh.cu, supported by MAEC-AECI

¶Instituto de Matemáticas, Universidad Nacional Autónoma de México, urrutia@matem.unam.mx, partially supported by grant FEDER-MTM2006-03909.

‖Departamento Matemática Aplicada II, Universidad de Sevilla, iventura@us.es, partially supported by grant FEDER-MTM2006-03909.

aligned boxes. In this paper we consider a convex set for the selection problem.

2 The largest monochromatic island

In this section we present an $O(n^3)$ time algorithm to solve the LMIP problem. Our algorithm is based on Fisher's $O(n^3 \log n)$ algorithm to solve the LMIP problem [17]. We improve Fisher's running time algorithm by using efficient data structures.

We proceed now to give some definitions that will be needed in the rest of this paper. $\triangle(p, e)$ will denote the triangle having a line segment e as one of its sides and the point p as the vertex opposite to e; $p_i \to p_j$ will denote the directed line segment starting at p_i and ending at p_j.

Let p_0 be a blue point in S, and S_0 the set of blue points in S below p_0. From now on we will assume that the elements of S_0 are labeled $\{p_1, \ldots, p_k\}$ in the counterclockwise order with respect to p_0.

Let $\mathcal{B}$ be a blue island of S, and p_0 the point of $\mathcal{B}$ with the largest y-coordinate (w.l.o.g. we will assume that such a point is unique). The point p_0 will be called the *anchor* of $\mathcal{B}$. The weight of $\mathcal{B}$ is the cardinality of $\mathcal{B}$.

Our approach to solve the LMIP is the following: For each blue point $p_0 \in S$, find the largest monochromatic island anchored in p_0.

Let $\mathcal{B}$ be a blue island anchored at p_0, and $p_0, p_{\sigma_1}, \ldots, p_{\sigma_k}$ be the vertices of the convex hull of $\mathcal{B}$ labeled in the counterclockwise order around the boundary of the convex hull of $\mathcal{B}$, starting at p_0. We call the edge $p_{\sigma_{k-1}} - p_{\sigma_k}$ joining $p_{\sigma_{k-1}}$ to p_{σ_k} the *last edge* of the convex hull of $\mathcal{B}$, and sometimes we will say that $\mathcal{B}$ *ends* at $p_{\sigma_{k-1}} - p_{\sigma_k}$. We denote as $\text{Blue}(P)$ the number of blue points contained in the convex set P. We will also assign a weight $w(p_{\sigma_{k-1}} - p_{\sigma_k})$ to the edge $p_{\sigma_{k-1}} - p_{\sigma_k}$ equal to the weight of the largest blue island of S anchored at p_0 that ends at $p_{\sigma_{k-1}} - p_{\sigma_k}$. If the triangle $\triangle(p_0, p_{\sigma_{k-1}} - p_{\sigma_k})$ contains at least one red point, $w(p_{\sigma_{k-1}} - p_{\sigma_k}) = 0$.

Let $\text{Blue}(p_{\sigma_i})$ denote the number of blue points inside or on the convex polygon with vertices $p_0, p_{\sigma_1}, \ldots, p_{\sigma_i}$, $(1 \leq i \leq k)$.

Our algorithm is based on the following crucial observation that suggests a dynamic programming approach to solve the problem:

Observation 1. $\text{Blue}(p_{\sigma i+1}) = \text{Blue}(p_{\sigma_i}) + \text{Blue}(\triangle(p_0, p_{\sigma_i} - p_{\sigma i+1})) - 2$.

The additive property in Observation 1, allows us to solve the problem of finding the largest monochromatic island anchored at a point p_0 by performing a radial sweep of the blue points below p_0 in the counterclockwise order around p_0 by joining sets of so-called *good triangles* with respect to p_0 [17]; that is sets of triangles with blue vertices (one of which is p_0), such that they have disjoint interiors, do not contain

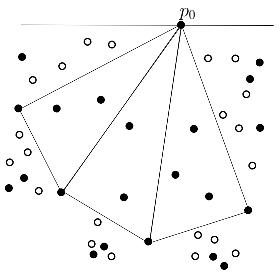


Figure 1: We add good triangles from left to right.

red points, and their union forms a convex polygon anchored at p_0, see Figure 1. The bottleneck of the sweeping approach proposed in [17] is the joining process. In the next section, we will show how to compute the LMIP for an anchored island in $O(n^2)$ time and space, thus obtaining an $O(n^3)$-time algorithm to solve the LMIP.

2.1 The algorithm

The algorithm does a preprocessing stage: First, we calculate *for all blue points* $p \in S$ the radial ordering of the blue points in S below p in overall $O(n^2)$ time and space [14] . Second, we preprocess S such that for each triangle with vertices in S, the number of red and the number of blue points in the triangle can be determined in constant time. We use a simple modification of the algorithm proposed in [15]. The preprocessing phase takes $O(n^2)$-time and space.

Let p_0 be a blue point in S, and relabel the blue points in S below p_0 by $\{p_1, \ldots, p_k\}$ such that the line segment joining p_0 to p_{i+1} is to the right of that joining p_0 to p_i. This labeling can be obtained in $O(n)$ time during the preprocessing stage. The algorithm scans $\{p_1, \ldots, p_k\}$ from p_1 to p_k such that for each two points p_i and p_j $(1 \leq i < j \leq k)$ we calculate the weight $w(p_i - p_j)$ to the edge joining p_i to p_j equal to the size of the largest blue island $\mathcal{B}$ (if any) of S anchored at p_0, and ending at the edge $p_i - p_j$. If the triangle $\triangle(p, p_i - p_j)$ contains at least one red point of S, $w(p_i - p_j)$ equals 0, see Figure 2. For the sake of clarity if $i < j$, we will orient the edge $p_i - p_j$ with the orientation $p_i \to p_j$, thus obtaining an oriented acyclic graph $G(p_0)$ with vertex set $\{p_1, \ldots, p_k\}$.

Observation 1 will allow us to calculate the weights $w(p_i \to p_j) = w(p_i - p_j)$ of the edges of $G(p_0)$ inductively as follows:

In the initial stage, when $i = 1$, we assign to all edges $p_1 \to p_j$ the weight $\text{Blue}(\Delta(p_0, p_1 - p_j))$, and $prev(p_1 \to p_j) = null$. Suppose that we have assigned $w(\cdot)$ and $prev(\cdot)$ to all edges $p_i \to p_j$ $(1 \leq j < k)$. We now show how to set the weights of all edges $p_j \to p_l$ $(j < l \leq k)$.

Assume that all incoming and outgoing edges to p_j are labelled $L_a = a_1, \ldots, a_q$ and $L_b = b_1, \ldots, b_r$

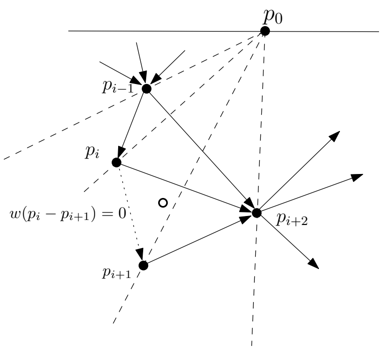


Figure 2: The weight of edge $p_i \to p_{i+1}$ is 0 because $\triangle(p_0, p_i \to p_j)$ contains a red point.

respectively such that L_a (resp. L_b) is sorted with respect to p_j, see Figure 3. During the preprocessing stage both L_a and L_b can be obtained in linear time. Given an edge a_r in L_a and an edge b_s in L_b, we say that a_r *is compatible* with b_s if the union of $\Delta(p_0, a_r)$ and $\Delta(p_0, b_s)$ is a convex polygon.

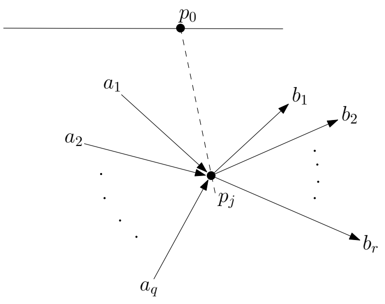


Figure 3: Ordering of the edges of p_j

Now, for every edge b_s in L_b such that triangle $\triangle(p_0, b_s)$ contains no red points in S, we want to find the edge a_r in L_a such that a_r is *compatible* with b_s and has maximum weight (if $\triangle(p_0, b_s)$ contains red points, then $w(b_s) = 0$). It is easy to see that if a_r exists then $w(b_s) = w(a_r) + \text{Blue}(\Delta(p_0, b_s)) - 2$. The index r will be $prev(b_s)$

To obtain r we proceed as follows: We label each a_i $(1 \le i \le q)$ with $max(a_i)$ which is a pointer to the edge a_h $(1 \le h \le i)$ such that $w(a_h)$ is maximum among $w(a_1), \dots, w(a_i)$. This can be done in $O(q)$ time by doing $max(a_1) = a_1$ and for $i = 2 \dots q$ applying the following formula:

$$max(a_i) = \begin{cases} a_i & \text{if } w(a_i) \ge w(max(a_{i-1})) \\ max(a_{i-1}) & \text{if } w(a_i) < w(max(a_{i-1})) \end{cases}$$

The following procedure computes $w(\cdot)$ and $prev(\cdot)$ for all b in L_b:

Start by setting $i = q$. For $m = 1, \dots, r$ find the first index t from i to 1 such that a_t is *compatible* with b_m. If such position t exists set $w(b_m) = w(max(a_t)) + \text{Blue}(\Delta(p_0, b_m)) - 2$ and $prev(b_m) = t$, otherwise set $t = 0$, $w(b_m) = \text{Blue}(\Delta(p_0, b_m))$ and $prev(b_m) = null$. After this make $i = t$ and continue the iteration of m.

Clearly above procedure runs in $O(n)$ time thus the complete assignment of $w(\cdot)$ and $prev(\cdot)$ is done in $O(n^2)$ time. Therefore we have proved:

Lemma 1 *Assigning $w(.)$ and $prev(.)$ to all edge of $G(p_0)$ can be done in $O(n^2)$ time.*

The largest blue island $\mathcal{B}$ anchored at p_0 corresponds to the edge e of $G(p_0)$ with maximum weight. Said in another way, $\mathcal{B}$ *ends* at e. The convex hull, and thus $\mathcal{B}$, can now be computed by using the pointers $prev(e)$ recursively.

By repeating the same procedure with the red points of S, we obtain the following result:

Theorem 2 *Let S be a bichromatic point set in general position on the plane. The largest monochromatic island can be found in $O(n^3)$ time, by using a preprocessing of $O(n^2)$ time and space.*

3 Generalizations

The algorithm presented in the previous section can be used to solve a collection of optimization problems. Suppose we have a function $f : \mathcal{P} \to I\!R$, where $\mathcal{P}$ is a set of convex polygons.

Definition 1 *We say that a function f on $\mathcal{P}$ [15] is decomposable iff for any polygon $P = \{p_1, p_2, \dots, p_k\}$ and any index $2 < i < k$,*

$$f(P) = g(f(\{p_1, \dots, p_i\}), f(\{p_1, p_i, \dots, p_k\}), p_1, p_i)$$

where g can be calculated in constant time.

Notice that if f counts the number of points of S contained within or on the boundary of a convex polygon P, then $g(x, y, p, q) = x + y - 2$.

Other decomposable functions are the perimeter, area, number of points in the convex hull, discrepancy, sum of weights of the points, etc. It is easy to see that our main algorithm can b e easily modified to minimizing or maximizing these functions. We can also modify the method for solving the problem of finding maximal or minimal empty polygons. Another interesting variant is the following:

The empty ordered heterochromatic island problem: *Given n points in the plane colored with colors $1, \dots, k$, compute the largest empty convex polygon P with k vertices such that the colors on the vertices of P appear in the order $1, \dots, k$.*

See [9] for a related paper. Note that the modification we have to do to our main algorithm is to build the graph $G(p_0)$ following colors in order $1, \dots, k$. The next result follows.

Theorem 3 *Let S be a set of points in the plane colored with k colors and let f be a monotone decomposable function. The ordered heterochromatic island that minimizes or maximizes f can be computed in $O(n^3)$ time and $O(n^2)$ space.*

4 The Class Cover Problem with Convex Sets

Given a set S of red and blue points, a classical problem is that known as the *Class Cover Problem* [7]. It consists in finding a set of circles with minimum cardinality such that every blue point in S is contained in at least one of the circles, and no circle contains a red point. We consider a variant of this problem in which we want to cover the blue points by using (non-necessarily) disjoint convex polygons. In [1] the problem of covering a point set with the smallest number of pairwise disjoint triangles is studied. It is proved by using a reduction from planar 3SAT-problem that this problem is NP-hard . We can use the same construction as in [1] to reduce any instance of the planar 3SAT to an instance of the Class Cover Problem with convex sets in which the only possible solutions are formed by sets of pairwise disjoint triangles. Hence our problem is also NP-hard.

The $O(\log n)$-approximation greedy approach for the more general *Set Cover Problem* [18] can easily be applied to our problem. It works as follows: recursively compute the maximum blue island I of S, remove it and repeat until there are no more blue points left in S. This approach produces a $O(n^4)$-time $O(\log n)$-approximation algorithm.

The techniques of [6] to obtain $o(\log n)$-approximation algorithms work for systems with constant VC-dimension. Notice that the VC-dimension of our system is not constant, specifically it can be $\Omega(n)$ (take a set of n points in convex position).

References

[1] P. K. Agarwal and S. Suri. *Surface Approximation and Geometric Partitions* Proc. 5th annual ACM-SIAM Symp. on Discrete Algorithms. ACM Press, 24-33.

[2] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. *Finding k points with minimum diameter and related problems.* Proc. 5th ACM Symp. on Computational Geometry, 283-291, 1989.

[3] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *Proceedings 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[4] D. Avis and D. Rappaport. *Computing the largest empty convex subset of a set of points.* In SCG 85: Proceedings of the first annual symposium on Computational geometry, 161167, ACM, 1985.

[5] J.E. Boyce, D.P. Dobkin, I. Robert L.(Scot) Drysdale, and L.J. Guibas. *Finding extremal polygons.* In STOC 82: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, 282289, ACM, 1982.

[6] H. Brönnimann, M.T. Goodrich. *Almost optimal set covers in finite VC-dimension.* Discrete and Computational Geometry, 14, 463-479, 1995.

[7] A.H. Cannon and L.J. Cowen. *Approximation algorithms for the class cover problem.* Annals of Mathematics and Artificial Intelligence, 40(3-4):215-223, 2004.

[8] T. H. Cormen, C. E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* Second Edition. MIT Press, McGraw-Hill, 2001.

[9] J.M. Díaz-Báñez, G. Hernandez, D. Oliveros, A. Ramirez-Vigueras, J.A. Sellarès, J. Urrutia, I. Ventura, Computing Shortest Heterochromatic Monotone Routes Operational Research Letters, Volume 36, 684-687, 2008.

[10] D.P. Dobkin, H. Edelsbrunner, and M.H. Overmars. *Searching for empty convex polygons* In SCG 88: Proceedings of the fourth annual symposium on Computational geometry, 224228, ACM, 1988.

[11] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *J. Computer and Systems Sciences*, 52(3) (1996) 453470.

[12] R. Duda, P. Hart, and D. Stork. *Pattern classification. John Wiley and Sons, Inc.*, 2001.

[13] J. Eckstein, P. L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its applications to data analysis. *Comput. Optim. Appl.* 23 (2002) 285–298.

[14] H. Edelsbrunner, J. O'Rourke, and R. Seidel. *Constructing arrangements of lines and hyperplanes with applications.* SIAM J. Comput. 15, 341-363, 1986.

[15] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. *Finding minimum area k-gons.* Discrete and Computational Geometry, 7, 4558, 1992.

[16] P. Erdös and G. Szekeres. *On some extremum problems in elementary geometry.* Ann. Univ. Sci. Budapest, (3-4), 5362, 1960/1.

[17] P. Fischer. *Sequential and parallel algorithms for finding a maximum convex polygon.* Computational Geometry: Theory and Applications, 7, 187200, 1997.

[18] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, NY. 1979.

On complexity of the mixed volume of parallelograms

Leonid Gurvits *

Abstract

Let $\mathbf{K} = (K_1...K_n)$ be a n-tuple of convex compact subsets in the Euclidean space R^n, and let $V(Cdot)$ be the Euclidean volume in R^n. The Minkowski polynomial $V_{\mathbf{K}}$ is defined as $V_{\mathbf{K}}(\lambda_1, ..., \lambda_n) = V(\lambda_1 K_1 + ... + \lambda_n K_n)$ and the mixed volume $V(K_1, ..., K_n)$ as

$$V(K_1, ..., K_n) = \frac{\partial^n}{\partial \lambda_1 ... \partial \lambda_n} V_{\mathbf{K}}(\lambda_1 K_1 + \cdots + \lambda_n K_n).$$

We study in this paper the seemingly simple case when each convex set K_i is a parallelogram, i.e. $K_i = \{aX_i + bY_i : 0 \leq a, b \leq 1; X_i, Y_i \in R^n$. If the number of distinct vectors in the set $|\{\frac{X_i}{||X_i||}, \frac{Y_i}{||Y_i||} : 1 \leq i \leq n\}|$ is equal to n then the mixed volume $V(K_1...K_n)$ can be computed as the permanent of $n \times n$ matrix having at most two nonzero entry in each column. Such permanent can be computed in deterministic polynomial time.
We prove that in general the mixed volume of parallelograms is $\#P-Complete$. On the positive side, we prove that any integer k there exists a deterministic poly-time algorithm approximating $V(K_1...K_n)$ with the relative accuracy $\frac{2^n}{n^k}$.

1 Introduction

Let $\mathbf{K} = (K_1...K_n)$ be a n-tuple of convex compact subsets in the Euclidean space R^n, and let $V(Cdot)$ be the Euclidean volume in R^n. It is well known Herman Minkowski result that the value of the $V_{\mathbf{K}}(\lambda_1 K_1 + \cdots + \lambda_n K_n)$ is a homogeneous polynomial of degree n, called the Minkowski polynomial, in nonnegative variables $\lambda_1...\lambda_n$, where $''+''$ denotes Minkowski sum, and λK denotes the dilatation of K with coefficient λ. The coefficient $V(K_1...K_n)$ of the monomial $\prod_{1 \leq i \leq n} \lambda_i$ is called the *mixed volume* of $K_1, ..., K_n$. Alternatively,

$$V(K_1, ..., K_n) = \frac{\partial^n}{\partial \lambda_1 ... \partial \lambda_n} V_{\mathbf{K}}(\lambda_1 K_1 + \cdots + \lambda_n K_n).$$

The problem of computing the mixed volume of convex bodies is also important for combinatorics and algebraic geometry [2]. For instance, the number of toric solutions to a generic system of n polynomial equations on C^n is equal (and in a general case is upper bounded by) to the mixed volume of the Newton polytopes of the the corresponding polynomials (see,for instance, [11]).

1.1 Previous Work

This remarkable connection, called **BKK Theorem**, created an "industry" of computing (exactly) the mixed volume of integer polytopes and its various generalizations, and most of algorithms in the area are of exponential runing time ([3], [4] and many more). Altough there was a substantial algorithmic activity on the mixed volume of polytopes prior to [2], the paper [2] was first, to our knowledge, systematic complexity-theoretic study in the area. It followed (naturally) famous FPRAS algorithms [1] for volumes of convex bodies , solved several natural problems and posed many important hard questions.The existence of FPRAS for the mixed volume even for polytopes or ellipsoids is still very open problem.
Efficient polynomial-time probabilistic algorithms that approximate the mixed volume extremely tightly ($(1+\epsilon)$-factor) were developed for some classes of well-presented convex bodies [2]. The algorithms in [2] are based on the multivariate polynomial interpolation and work if and only if the number k of distinct convex sets in the tuple $\mathbf{K}$ is "small", i.e. $k = O(\log(n))$. It was proved in the recent paper [8] that there exists a randomized poly-time algorithm approximating the mixed volume of n well-presented compact convex sets in R^n with relative accuracy $\frac{n^n}{n!} \approx e^n$. If the convex sets are presented by oracles, then, because of **Barany-Furedi bound**, even such accuracy can not be achieved by a deterministic poly-time algorithm.
In the case of small affine dimension of the convex sets the algorithm from [8] gives a better exponent. For instance, if the affine dimensions $aff(K_i) \leq 2 : 3 \leq i \leq n$ then the corresponding relative accuracy is bounded by $(1 + \sqrt{2})^n$.

2 The mixed volume of parallelograms in R^n

Let $\mathbf{X} = \{X_1, ..., X_k\} \subset R^n$ be a finite subset of real n-dimensional vectors. A *zonotope* $Zon_{\mathbf{X}}$ is defined as

$$Zon_{\mathbf{X}} = \{ \sum_{1 \leq i \leq k} a_i X_i : 0 \leq a_i \leq 1, 1 \leq i \leq k\}$$

A *zonotope* $Zon(X_1, X_2) =: \{aX_1 + bX_2 : 0 \leq a, b \leq$

*gurvits@lanl.gov. Los Alamos National Laboratory, Los Alamos, NM.

$1\}$ is called *parallelogram*. If the mixed volume

$$MV =: V(K_1, ..., K_n) \neq 0, K_i = Zon(X_{i,0}, X_{i,1}),$$

then we can assume, modulo poly-time preprocessing, that $\{X_{i,0} = e_i, i \leq i \leq n\}$ is the standard basis in R^n. Define a $n \times n$ matrix $A = [Y_1|...|Y_n], Y_i = X_{i,1}$ and PSD rank two matrices $Q_i = e_i e_i^T + Y_i Y_i^T$.
Then the mixed volume

$$MV =: MV(A) = \sum_{S \subset \{1,...,n\}} |\det(A_{S,S})|$$

and the mixed discriminant

$$D(Q_1, ..., Q_n) =: D(A) = \sum_{S \subset \{1,...,n\}} |\det(A_{S,S})|^2,$$

where $A_{S,S}$ is the corresponding principal submatrix of A. If the matrix A is *principally unimodular*, i.e. $|\det(A_{S,S})| \in \{0, 1\}$, then $MV(A) = D(A)$.
Note that

$$D^{\frac{1}{2}} \leq MV \leq 2^{\frac{n}{2}} D^{\frac{1}{2}} \qquad (1)$$

3 #P-Completeness

3.1 Universality of the mixed volume of (special) parallelograms and the mixed discriminant of (special) rank two PSD matrices

Given k pairs of vectors $(X_i, Y_i) \in R^n : 1 \leq i \leq k$ we define the generalized mixed discriminant

$$GD((X_1, Y_1), ..., (X_k, Y_k)) = \\ = \sum_{1 \leq j_1 < ... < j_n \leq k} |\det[X_{j_1}|...|X_{j_n}]|^2 |\det[y_{j_1}|...|Y_{j_n}]|^2 \qquad (2)$$

and the generalized mixed volume

$$GV((X_1, Y_1), ..., (X_k, Y_k)) = \\ = \sum_{1 \leq j_1 < ... < j_n \leq k} |\det[X_{j_1}|...|X_{j_n}]||\det[y_{j_1}|...|Y_{j_n}]| \qquad (3)$$

The formula (3) generalizes the mixed volume of zonotopes $V(Zon_{\mathbf{X}_1}, ..., Zon_{\mathbf{X}_n})$, the formula (2) generalizes the mixed discriminant of positive semidefinite tuples $D(PSD_{\mathbf{X}_1}, ..., PSD_{\mathbf{X}_n})$. In these both cases the corresponding pairs of vectors are $\{e_i, Y_i) : Y_i \in \mathbf{X}_i; 1 \leq i \leq n\}$ and $\{e_1, ..., e_n\}$ is the standard basis in C^n.

Definition 1 *Consider k vectors $V_1, ..., V_k \in R^n, n \leq k$ and k vectors $Z_1, ..., Z_k \in R^{k-n}$. We define $X_i = V_i \oplus 0 \in R^k, W_i = 0 \oplus Z_i \in R^k$ and correspondingly*

$$A_i = X_i X_i^* + W_i W_i^* = V_i V_i^* \oplus Z_i Z_i^*, P_i = Zon(X_i, W_i).$$

The tuples of block-diagonal rank two PSD matrices $(A_1, ..., A_k)$ and parallelograms $(P_1, ..., P_k)$ are called direct sum rank two tuples.

3.2 Dual Geometric Matroids

Consider $n \times k$ matrix
$\{M(i,j) : 1 \leq i \leq n, 1 \leq j \leq k\} = M = [A|B]$ over an arbitrary commutative field. We assume that $k \geq n$, and $n \times n$ matrix A is nonsingular. Notice that $[A|B] = A[I|A^{-1}B]$.
Let C be an $k-n \times k-n$ matrix such that $\det(C) = \det(A)$. Define the following $k-n \times k$ matrix over the same field:

$$Dual(M) = C[B^T(A^{-1})^T|I] \qquad (4)$$

Proposition 1 *Consider a subset $S = \{1 \leq i_i < ... < i_n \leq k\} \subset \{1, ..., k\}$. Define the following $n \times n$ (sub)matrix $M_S = \{M(i,j) : 1 \leq i \leq n, j \in S\}$ and the $n-k \times n-k$ (sub)matrix $Dual(M)_{S'} = \{Dual(M)(k,l) : 1 \leq k \leq n-k, l \in S'$. Here the subset $S' = \{1, ..., k\} - S$ is the compliment of S. Then*

$$\det(M_S) = \pm \det(Dual(M)_{S'}) \qquad (5)$$

For a proof, consider WLOG the case $A = I, C = I$.

3.3 The reduction to *direct sum* rank two case

Theorem 2 *1. Let k pairs of vectors $(X_i, Y_i) \in R^n : 1 \leq i \leq k$ be given. Then there is a deterministic polynomial time algorithm (preprocessing) which outputs a direct sum rank two matrix tuple $\mathbf{A} = (A_1, ..., A_k)$ such that*

$$GD((X_1, Y_1), ..., (X_k, Y_k)) = D(A_1, ..., A_k) \quad (6)$$

The Generalized Mixed Discriminant can be represented as the Mixed Discriminant of a *direct sum* rank two tuple.

2. Let k pairs of vectors $(X_i, Y_i) \in R^n : 1 \leq i \leq k$ be given. Then there is a deterministic polynomial time algorithm (preprocessing) which outputs aspecial rank two parallelogram tuple $\mathbf{P} = (P_1, ..., P_k)$ such that

$$GV((X_1, Y_1), ..., (X_k, Y_k)) = V(P_1, ..., P_k) \quad (7)$$

The Generalized Mixed Volume can be represented as the Mixed Volume of a *direct sum* rank two tuple of parallelograms.

Proof. Define the following $n \times k$ matrix $M = [Y_1|...|Y_k]$. If $Rank(M) < n$ then $GD((X_1, Y_1), ..., (X_k, Y_k)) = GV((X_1, Y_1), ..., (X_k, Y_k)) = V(P_1, ..., P_k) = 0$.
If $Rank(M) = n$ then, up to some permutations $\pi \in S_k$ of columns, $M = [A|B] : A$ is $n \times n$ nonsingular matrix. Obviously, such block-representation can be found in poly-time; also the $(k-n) \times k$ matrix $Dual(M)$, defined in (4), can be constructed in poly-time. Let $Z_1, ..., Z_k \in R^{k-n}$ be the ordered

columns of $Dual(M)$. Define, as in Definition (1), *direct sum* rank two parallelogram tuple
$P_i = Zon(X_i, W_i) \subset R^k : X_i = V_i \oplus 0 \in R^k, W_i = 0 \oplus Z_i \in R^k; 1 \leq i \leq k$.
It follows now from (5) that
$GV((X_1, Y_1), ..., (X_k, Y_k)) = V(P_1, ..., P_k)$. The mixed discriminant statement (6) follows in the similar way. $\square$

Let $A = I + M(\pi_1) + M(\pi_2)$ be $n \times n$ boolean matrix with exactly three ones in each row and column. Here π_1, π_2 are two non-overlaping permutations without fixed points, $M(\pi_i), i = 1, 2$ are the corresponding permutation matrices. The next Lemma(3) is a rather direct corollary of Theorem (2), it corresponds to the following $3n$ pairs of vectors:
$\{(e_i, e_{\sigma(i)}) : 1 \leq i \leq n; \sigma \in \{Id, \pi_1, \pi_2\}\}$.

Lemma 3 *Consider the following $3n \times 3n$ matrix:*

$$NPM = \begin{pmatrix} 0 & I & I \\ (M(\pi_1))^T & 0 & 0 \\ (M(\pi_2))^T & 0 & 0 \end{pmatrix}$$

Then the matrix NPM is principally unimodular ; the permanent

$$Per(A) = MV(NPM) = D(NPM).$$

The permanent $Perm(I + M(\pi_1) + M(\pi_2))$ is equal to the number of perfect matchings in the corresponding 3-regular bipartite graph. Therefore (see, for instance, [10]) computing $Per(G)$ is $\#P - Complete$. The next result follows now directly.

Theorem 4 *As the mixed volume of parallelograms $MV(A)$ as well the mixed discriminant of rank two PSD matrices are $\#P - Complete$ even for very sparse (at most two ones in each row and column) boolean principally unimodular matrices.*

(The #P-Completeness of the (mixed discriminant of rank two PSD matrices) $MD(A)$ was proved in [6]. The proof in [6] is much more complicated comparing with the present paper, more importantly it does not apply to the principally unimodular matrices.)

4 Deterministic poly-time approximation algorithms

4.1 A few words on randomized algorithm from [8]

A randomized algorithm from [8] based on the next inequality:

Theorem 5 *Let $\mathbf{K} = (K_1 ... K_n)$ be a n-tuple convex compact subsets in the Euclidean space R^n and $aff(i)$ be the affine dimension of K_i , $1 \leq i \leq n$. Devine the capacity of a Minkowski volume polynomial $V_{\mathbf{K}}$ as*

$$Cap(V_{\mathbf{K}}) = \inf_{x_i > 0 : 1 \leq i \leq n} \frac{V_{\mathbf{K}}(x_1, ..., x_n)}{\prod_{1 \leq i \leq n} x_i}.$$

Define the next function $\lambda(n, k)$:
$\lambda(n, k) = (\min_{x > 0}(\frac{sv_{n,k}(x)}{x}))^{-1}$,
$sv_{n,k}(x) = 1 + \sum_{1 \leq i \leq k} (\frac{x}{n})^i nChoosei; n, k \geq 1$.

Then the following inequality holds:
$Cap(V_{\mathbf{K}}) \geq V(K_1, ..., K_n) \geq$
$\geq \prod_{1 \leq i \leq n} \lambda(i, \min(i, aff(i))) Cap(V_{\mathbf{K}})$

As the affine dimension of parallelograms is at most 2, we get for tuples of parallelograms that

$$Cap(V_{\mathbf{K}}) \geq V(K_1, ..., K_n) \geq \frac{1}{2}(1 + \sqrt{2})^{2-n} Cap(V_{\mathbf{K}}) \tag{8}$$

So, in order to get $(1 + \sqrt{2})^n$ relative accuracy it is sufficient to approximate $\log(Cap(V_{\mathbf{K}}))$ with an addtive error $O(1)$. $\log(Cap(V_{\mathbf{K}}))$ is a solution of the next convex minimization problem:

$$\log(Cap(V_{\mathbf{K}})) = \inf_{y_1 + ... + y_n = 0} \log(V_{\mathbf{K}}(e^{y_1}, ..., e^{y_n})). \tag{9}$$

To solve this convex problem we need an oracle evaluating $\log(V_{\mathbf{K}}(e^{y_1}, ..., e^{y_n}))$ with small additive error. Unfortunatelly, it was proved in [2] that computing the volume of a Minkowski sum of parallelograms is #P-Complete. Therefore it is not clear whether a randomized algorithm from [8] can be derandomized.

4.2 Another failed approach

In order to get a 2^n-approximation algorithm it would be sufficient to compute $\max_S |\det(A_{S,S}|$. It was proved in [6] that this maximization problem is **NP-Complete**.

4.3 H-Stable polynomials

A homogeneous polynomial $p(x_1, ..., x_n)$ with nonnegative coefficients is called **H-Stable** if $p(z_1, ..., z_n) \neq 0$ provided that the real parts $Re(z_i) > 0, 1 \leq i \leq n$.

Let $\mathbf{A} = (A_1, ..., A_n)$ be an n-tuple of $n \times n$ complex hermitian PSD matrices; the corresponding determinantal polynomial is defined as $Det_{\mathbf{A}}(\lambda_1, ..., \lambda_n) = \det(\sum_{1 \leq i \leq n} \lambda_i A_i)$. We benefit in this paper from the fact that the polynomial $Det_{\mathbf{A}}$ is **H-Stable** if it is not zero, i.e. if the sum $\sum_{1 \leq i \leq n} A_i \succ 0$ (is positive definite). It is easy to see that the degree $deg_{Det_{\mathbf{A}}}(i)$ of the variable λ_i in the polynomial $Det_{\mathbf{A}}$ is equal $deg_{Det_{\mathbf{A}}}(i) = Rank(A_i)$.

We need the next general result from [9]:

Theorem 6 *Let p be* **H-Stable** *polynomial with nonnegative coefficients in n variables and of degree*

n. Define $G(k) =: (\frac{k-1}{k})^{k-1}, k \geq 1$. Then the following inequalitities hold:
$Cap(p) \geq \frac{\partial^n}{\partial x_1 ... \partial x_n} p(0, ..., 0) \geq$
$\geq \prod_{2 \leq i \leq n} G\left(\min(i, deg_p(i))\right) Cap(p)$.

Theorem 7 *Let $\mathbf{A} = (A_1, ..., A_n)$ be an n-tuple of $n \times n$ complex hermitian PSD matrices. Assume that $Rank(A_i) \leq 2 : 1 \leq i \leq n - [k \log(n)]$, where k is a fixed integer. Then there exists a deterministic poly-time algorithm approximating the mixed discriminant $D(A_1, ..., A_n)$ with the relative accuracy $\frac{2^n}{n^k}$.*

Proof. [Sketch] Assume WLOG that the mixed discriminant $D(A_1, ..., A_n) > 0$ and the tuple $\mathbf{A} = (A_1, ..., A_n)$ is indecomposable as defined in [5], [7].

Define the next polynomial in $l =: n - [k \log(n)]$ variables and of degree l:

$$q(x_1, ..., x_l) = \frac{\partial^{n-l}}{\partial x_{l+1} ... \partial x_n} Det_{\mathbf{A}}(x_1, ..., x_l, 0, ..., 0).$$

Then q is also **H-Stable**(see [9]) and $deg_q(i) \leq 2, 1 \leq i \leq l$. Note that

$$D(A_1, ..., A_n) = \frac{\partial^l}{\partial x_1 ... \partial x_l} q(0, ..., 0).$$

It follows from Theorem(6) that

$$1 \leq \frac{Cap(q)}{D(A_1, ..., A_n)} \leq 2^{l-1} \approx \frac{2^n}{n^k}.$$

We need to compute now $\log(Cap(q))$ with an additive error $O(1)$. Since the tuple $\mathbf{A} = (A_1, ..., A_n)$ is indecomposable hence the polynomial q is indecomposable as defined in [7]:

$$\frac{\partial^l}{\partial x_i \partial x_j \prod_{m \neq (i,j)} \partial x_m} q(0, ..., 0) > 0 : 1 \leq i \neq j \leq l. \tag{10}$$

The condition (10) guaranties that convex minimization problem

$$\min_{y_1 + ... + y_n = 0} \log(q(e^{y_1}, ..., e^{y_n}))$$

has unique solution and the minimum value can be approximated with an additive error $O(1)$ using polynomial number of evaluations of the polynomial q. Finally, we remark that to exactly evaluate $q(x_1, ..., x_l)$ one needs to compute just $O(n^{k+1})$ determinants of some effectively computable $n \times n$ matrices. Therefore there exists a number $F(k) = exp(\log(Cap(q) + O(1))$, computable in deterministic poly-time, such that

$$1 \leq \frac{D(A_1, ..., A_n)}{F(k)} \leq \frac{2^n}{n^k}. \tag{11}$$

□

Theorem 8 *Given a tuple of parallelograms*

$$(K_1, ..., K_n); K_i = \{aX_i + bY_i : 0 \leq a, b \leq 1; X_i, Y_i \in R^n\}, 1 \leq i \leq n$$

and integer number k, there exists a number $H(k)$, computable in deterministic poly-time, such that

$$1 \leq \frac{MV}{H(k)} \leq \frac{2^n}{n^{\frac{k}{2}}}.$$

Proof. Consider rank 2 PSD matrices $A_i = X_i X_i^T + Y_i Y_i^T$ and the number $F(k)$ from (11). It follows from the inequality (1) that $1 \leq \frac{MV}{F(k)^{\frac{1}{2}}} \leq \frac{2^n}{n^{\frac{k}{2}}}$. Therefore, we get for the number $H(k) = F(k)^{\frac{1}{2}}$ that $1 \leq \frac{MV}{H(k)} \leq \frac{2^n}{n^{\frac{k}{2}}}$.

□

References

[1] M. Dyer and A. Frieze, The complexity of computing the volume of a polyhedron, *SIAM J. Comput.*, 17, 967-994, 1988.

[2] M. Dyer, P. Gritzmann and A. Hufnagel, On the complexity of computing mixed volumes, *SIAM J. Comput.*, 27(2), 356-400, 1998.

[3] I.Z. Emiris and J.F. Canny, Efficient incremental algorithms for sparse resultant and the mixed volume, Journal of Symbolic Computation, 20, 117-149, 1995.

[4] T. Gao, T.Y. Li and M. Wu, MixedVol: A software package for Mixed Volume computation, ACM Transactions on Math. Software 31(4), 2005.

[5] L.Gurvits and A. Samorodnitsky, A deterministic algorithm approximating the mixed discriminant and mixed volume, and a combinatorial corollary, *Discrete Comput. Geom.* 27: 531 -550, 2002.

[6] L. Gurvits, On the Complexity of Mixed Discriminants and Related Problems. MFCS 2005: 447-458.

[7] L. Gurvits, Hyperbolic Polynomials Approach to Van der Waerden/Schrijver-Valiant like Conjectures : Sharper Bounds , Simpler Proofs and Algorithmic Applications , in Proc. of STOC-2006 , 2006.

[8] L. Gurvits, A polynomial time algorithm to approximate the mixed volume within a simply exponential factor, available at http://arxiv.org/abs/cs/0702013.

[9] L. Gurvits, Van der Waerden/Schrijver-Valiant like Conjectures and Stable (aka Hyperbolic) Homogeneous Polynomials : One Theorem for all, Electron. J. Combin. 15 (2008), no. 1, Research Paper 66, 26 pp.

[10] P. Dagum and M. Luby, Approximating the permanent of graphs with large factors. Theoretical Computer Science, 102:283– 305, 1992.

[11] Bernd Sturmfels, Polynomial equations and convex polytopes. Amer. Math. Monthly 105 (1998), no. 10, 907–922

Cache-Oblivious Construction of a Well-Separated Pair Decomposition

Fabian Gieseke[*] Jan Vahrenhold[†]

Abstract

We present a cache-oblivious algorithm for computing a well-separated pair decomposition of a finite point set $S \subset \mathbb{R}^d$ using $\mathcal{O}(\text{sort}(|S|))$ memory transfers.

1 Preliminaries

Given two point sets $A, B \subset \mathbb{R}^d$ with bounding hyperrectangles $R(A)$ and $R(B)$ and a real constant $s > 0$, we say that A and B are *well-separated* with respect to s if there are two disjoint balls $B_{(a,r)}$ and $B_{(b,r)}$ with radius $r \geq 0$ and centers $a, b \in \mathbb{R}^d$ such that $R(A) \subset B_{(a,r)}, R(B) \subset B_{(b,r)}$ and the distance between $B_{(a,r)}$ and $B_{(b,r)}$ is at least sr. We refer to s as the *separation ratio*. Callahan and Kosaraju [7] defined a *well-separated pair decomposition* (WSPD) for the point set S with separation ratio $s > 0$ as a sequence $\{\{A_1, B_1\}, \ldots, \{A_m, B_m\}\}$ of pairs of non-empty subsets of S such that

1. $A_i \cap B_i = \emptyset$ for all $i = 1, \ldots, m$,
2. for each pair $\{p, q\}$ of distinct points of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that $p \in A_i$ and $q \in B_i$ or vice versa,
3. A_i and B_i are well-separated with respect to s for all $i = 1, \ldots, m$.

The integer m is called the *size* of the WSPD.

Observation 1 ([7]) *Let S be a set of points in $\mathbb{R}^d$ and let $s > 0$ be a real number. A WSPD for S with respect to s having size $\mathcal{O}(s^d|S|)$ can be computed in $\mathcal{O}(|S|\log|S| + s^d|S|)$ time.*

For the analysis in this paper we use the *cache-oblivious model* [8], which is a variant of the standard two-level *I/O model* introduced by Aggarwal and Vitter [1]. Their model defines N to denote the number of objects in the problem instance, $M \ll N$ to denote the number of objects fitting into internal memory, and $2 \leq B \leq M/2$ to denote the number of objects per disk block. In the *cache-oblivious model* of Frigo *et al.* [8], we also have the parameters N, M, and B for the analysis of an algorithm, but in the design-phase of an algorithm, the parameters of M and B must not be used. This allows us to model a multi-level hierarchical memory, where memory is transferred in blocks between any two adjacent layers of memory (and not only between memory and disk). Frigo *et al.* [8] showed that sorting N items requires $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ memory transfers while scanning N items can obviously be done in $\Theta(\frac{N}{B})$ memory transfers; both bounds match the I/O-bounds in the I/O-model. We will use the shorthands $\mathcal{O}(\text{sort}(X)) = \mathcal{O}(\frac{X}{B}\log_{M/B}\frac{X}{B})$ and $\mathcal{O}(\text{scan}(X)) = \mathcal{O}(\frac{X}{B})$.

2 Cache-Oblivious WSPD

We show that we can efficiently construct a WSPD in the cache-oblivious model of computation using only $\mathcal{O}(\text{sort}(|S|))$ memory transfers. The algorithm follows the I/O-efficient approach of Govindarajan *et al.* [10]:

Observation 2 ([10]) *Given a set S of points in $\mathbb{R}^d$ and a real constant $s > 0$, a WSPD for S with separation ratio s and size $\mathcal{O}(s^d|S|)$ can be computed in $\mathcal{O}(\text{sort}(|S|))$ I/Os using $\mathcal{O}(|S|/B)$ disk blocks.*

Most of their algorithm is based on scanning and sorting or on other techniques which have a cache-oblivious counterpart. However, some parts of their algorithm use the values of M and B (which are unknown to a cache-oblivious algorithm) or take advantage of data structures with no (direct) cache-oblivious counterpart. In particular, Govindarajan *et al.* use two such data structures, namely *buffer trees* [2] and *topology buffer trees* [12]. Replacing them by efficient cache-oblivious computation steps constitutes a major part of our modifications. We describe our cache-oblivious method by outlining the I/O-efficient algorithm along with the necessary modifications.[1]

In a nutshell, Govindarajan *et al.* [10] compute a WSPD for S as follows: They start by constructing a special binary tree T, called *fair split tree* of S, whose nodes are used to represent the pairs of the WSPD and whose leaves are in a one-to-one correspondence with the points in S. The idea is to build T recursively. First, a "partial" fair split tree T' with leaves of bounded *sizes*, i.e., whose leaves correspond to at most $|S|^\alpha$ points for some constant $1 - \frac{1}{2d} \leq \alpha < 1$, is constructed. They then recursively build the split trees for the leaves, proceeding with the optimal internal memory algorithm for every leaf whose size is at most M. After the computation of the split tree they

[*]Faculty of Computer Science, LS XI, TU Dortmund, 44227 Dortmund, Germany, `fabian.gieseke@cs.tu-dortmund.de`.

[†]Faculty of Computer Science, LS XI, TU Dortmund, 44227 Dortmund, Germany, `jan.vahrenhold@cs.tu-dortmund.de`

[1]See the thesis of Gieseke [9] for all details.

Algorithm 1 Construction of a fair split tree.

```
Function FairSplitTree($S, R_0$)
Require: A point set $S \subset \mathbb{R}^d$ and a box $R_0$ with $S \subset R_0$
Ensure: A fair split tree $T$ of $S$
1: if $|S| = 1$ then
2:     Construct a single-node fair split tree.
3: else
4:     $T' :=$ PartialFairSplitTree($S, R_0$).
       (Note: For each leaf $S_1, \ldots, S_k$ of $T'$: $|S_i| \leq |S|^\alpha$)
5:     for $i = 1$ to $k$ do
6:         $T_i :=$ FairSplitTree($S_i, \hat{R}(S_i)$).
           (Note: $\hat{R}(S_i)$ denotes an outer rectangle, which
           is box fulfilling certain properties)
7: return $T := T' \cup T_1 \cup \cdots \cup T_k$
```

simulate the internal memory algorithm of Callahan and Kosaraju [7] for constructing the pairs $\{A_i, B_i\}$ in external memory using time-forward processing [2].

2.1 Construction of a Fair Split Tree

To compute T, we invoke FairSplitTree with the point set S and a centered minimal bounding cube R_0 containing S. The structure and the analysis of function FairSplitTree are close to those of the original I/O-efficient algorithm. However, we do not stop the recursion at leaves of size M (which are unknown to a cache-oblivious algorithm). Also, we set the constant α to $\alpha \in [1 - \frac{1}{4d}, 1)$, thus effectively slowing down the recursion. In Lemma 2 it will be shown that the function PartialFairSplitTree computes a valid partial fair split tree T' of S. Thus, the overall algorithm correctly computes the desired split tree T of S performing $\mathcal{I}(|S|) \leq c \cdot \text{sort}(|S|) + \sum_{i=1}^{k} \mathcal{I}(|S_i|) = \mathcal{O}(\text{sort}(|S|))$ memory transfers. The linear bound for the space consumption follows from the fact that the function PartialFairSplitTree uses $\mathcal{O}(|S|/B)$ space and from the fact that $\sum_{i=1}^{k} |S_i| = |S|$.

Lemma 1 *Given a set S of points in $\mathbb{R}^d$, function* FairSplitTree *(Algorithm 1) computes a fair split tree of S in $\mathcal{O}(\text{sort}(|S|))$ memory transfers using $\mathcal{O}(|S|/B)$ blocks of external memory.*

2.2 Construction of a Partial Fair Split Tree

It remains to show that PartialFairSplitTree (Algorithm 2) causes $\mathcal{O}(\text{sort}(|S|))$ memory transfers and uses $\mathcal{O}(|S|/B)$ blocks of memory. The framework of this function is identical to that of the original I/O-efficient algorithm for computing the partial fair split tree T'. In the first step, a binary tree T_c, called *compressed pseudo split tree*, is computed whose nodes correspond to *boxes*, i.e. hyperrectangles R that fulfill the inequality $l_{\max}(R) \leq 3 \cdot l_{\min}(R)$, where the latter notations denote the maximum and minimum length of R, respectively. In the second step, the tree T_c is expanded to a tree T'', called a *pseudo split tree*. In the last step, leaves which do not contain any points of S are removed and paths consisting of nodes having one child are compressed to single edges. The resulting tree is the desired partial fair split tree T' of S.

Algorithm 2 Construction of a partial fair split tree.

```
Function PartialFairSplitTree($S, R_0$)
Require: A point set $S \subset \mathbb{R}^d$ and a box $R_0$ with $S \subset R_0$
Ensure: A partial fair split tree $T'$ of $S$ whose leaves have
    size at most $|S|^\alpha$
1: Compute a compressed pseudo split tree $T_c$ of $S$, where
   every node represents a box.
2: Expand $T_c$ to the pseudo split tree $T''$.
3: Remove every node of $T''$ that does not contain any
   points of $S$ and compress every path of nodes having
   one child into a single edge to obtain $T'$.
4: return $T'$
```

Govindarajan *et al.* first partition every dimension of the starting hyperrectangle R_0 into slabs each containing at most $|S|^\alpha$ points. For each dimension, these slabs are bounded by $\left\lceil |S|^{1-\alpha} \right\rceil + 1$ axis-parallel hyperplanes, called the *slab boundaries*. The two extreme slab boundaries in each dimension are positioned in such a way that they contain the two sides of the starting hyperrectangle R_0 in this dimension. The construction of the tree T_c is only based on the information provided by these slab boundaries and proceeds by splitting a hyperrectangle R fulfilling specific invariants into one or two smaller hyperrectangles fulfilling these invariants as well.

These invariants guarantee that each hyperrectangle side obtained by this splitting process can be described uniquely using a constant number of slab boundaries and a constant amount of additional information. As this is also true for the starting hyperrectangle R_0 (each of its sides is contained in a slab boundary), any hyperrectangle that can be obtained from R_0 through a sequence of splits can be described using a constant number of slab boundaries and a constant amount of extra information. The main idea of the I/O-efficient algorithm of Govindarajan *et al.* is to construct the set of *all* hyperrectangles that can be described this way. Callahan [5] calls these hyperrectangles *constructible* and bounds theirs total number:

Observation 3 ([5, 10]) *There are $\mathcal{O}(|S|^{2d(1-\alpha)})$ constructible hyperrectangles.*

Hence, by setting the value for the constant $\alpha \in [1 - \frac{1}{2d}, 1)$, Govindarajan *et al.* get $\mathcal{O}(|S|)$ constructible hyperrectangles. Note that, due to our minor, yet crucial modification of the value of α, we can bound the number of all constructible hyperrectangles by $\mathcal{O}(\sqrt{|S|})$. Ultimately, this will allow us to replace both the use of buffer trees as well as the use of topology buffer trees by efficient cache-oblivious computation steps. Govindarajan *et al.* then construct a graph Γ with nodes corresponding to the set of constructible hyperrectangles and with an edge set originating from the splitting process mentioned above. The tree T_c consists of all nodes of Γ that are accessible from R_0 along with their outgoing edges. The two main steps of the I/O-efficient construction of T_c are (a) building the graph Γ in a preprocessing step and (b) extracting the tree T_c afterwards.

Step 1a: Building $\mathbf{\Gamma}$ Constructing the slab boundaries is based on sorting and scanning, and a cache-oblivious implementation uses $\mathcal{O}(d \cdot \text{sort}(|S|)) = \mathcal{O}(\text{sort}(|S|))$ memory transfers. The vertex set of Γ, i.e. the set of all constructible hyperrectangles, can be constructed based on the information given by the slab boundaries. Govindarajan *et al.* do this by performing $\mathcal{O}(d)$ nested scans using $\mathcal{O}(\text{scan}(|\Gamma|))$ I/Os each. They then use buffer trees [2] over the appropriate list of slab boundaries and answer I/O-efficient search queries on the constructed trees to augment every vertex R with a description of the largest hyperrectangle R' contained in R and bounded by slab boundaries as well as with a description of the slab boundary that comes closest to the “middle” of R in each dimension. They use this information to construct the edge set of Γ by a single scan of Γ's vertices. Thus, the construction of Γ uses $\mathcal{O}(\text{sort}(|S|))$ I/Os.

The vertex set of Γ can be constructed in a cache-oblivious manner using $\mathcal{O}(d)$ nested scans as well. Instead of using buffer trees, we apply a brute-force approach for augmenting the nodes of Γ with the appropriate information. This approach simply scans the (sorted) list of slab boundaries in the i-th dimension to find the appropriate slab boundaries in each dimension. Due to our choice of α, each vertex of Γ can be processed this way using an overall number of $\mathcal{O}(d \cdot \text{scan}(|S|^{1-\alpha})) = \mathcal{O}(\text{scan}(\sqrt{|S|}))$ memory transfers. Thus, the overall process for all $\mathcal{O}(\sqrt{|S|})$ vertices of Γ requires $\mathcal{O}(\text{scan}(|S|))$ memory transfers.

The final step, to construct the edge set of Γ, can be done analogously by a single scan over the vertex set of Γ taking $\mathcal{O}(\text{scan}(|\Gamma|)) = \mathcal{O}(\text{scan}(\sqrt{|S|}))$ memory transfers. This completes the construction of Γ.

Step 1b: Extracting $\mathbf{T_c}$ from $\mathbf{\Gamma}$ The I/O-efficient extraction of T_c from Γ can be implemented using time-forward processing and topological sorting. These steps can be performed efficiently in the cache-oblivious model [3, 4], and thus the extraction of T_c can be done the same way by spending $\mathcal{O}(\text{sort}(|\Gamma|))$ memory transfers. With our choice of α, this step takes $\mathcal{O}(\text{sort}(\sqrt{|S|}))$ memory transfers.

Step 2: Constructing $\mathbf{T''}$ Given the compressed pseudo split tree T_c for S, Govindarajan *et al.* construct the pseudo split tree T'' in three phases: In the first phase, they attach some leaves to T_c which were discarded during the construction of T_c. As at most one child is attached to each node of T_c, the resulting graph, called T_c^+, has size $\mathcal{O}(|T_c|)$ as well. This step can be implemented cache-obliviously in a straightforward manner (see [9] for the details) and takes $\mathcal{O}(\text{scan}(\sqrt{|S|}))$ memory transfers.

To distribute the points of S to the proper boxes and regions, Govindarajan *et al.* answer so-called *deepest containment queries* on T_c^+ for all points in S using a topology buffer tree [12]. The distribution of all points takes $\mathcal{O}(\text{sort}(|S|))$ I/Os.

Instead of using a topology buffer tree, we apply the following brute-force approach for distributing the points : Subdivide the space into cells bounded by the hyperplanes which occur as boundaries for the constructible hyperrectangles. Using the same argument as for bounding the number of constructible hyperrectangles ensures that there are only $\mathcal{O}(|S|^{2d(1-\alpha)}) = \mathcal{O}(\sqrt{|S|})$ cells of this type [5, 10]. Furthermore, a list of these cells can be constructed using $\mathcal{O}(d)$ nested scanning and sorting steps using $\mathcal{O}(\text{sort}(\sqrt{|S|}))$ memory transfers. By sorting the points and the list of hyperplanes with respect to the ith coordinate and by subsequently scanning both sorted lists, we can label each point with the pair of adjacent hyperplanes in this dimension, which determines the slab that contains it. Processing all points and all d dimensions this way takes $\mathcal{O}(\text{sort}(|S|))$ memory transfers and labels each point with a description of the cell that contains it. After computing this correspondence between the points in S and the cells, we compute an appropriate correspondence between the cells and the hyperrectangles and regions. By construction, each cell is either contained in a hyperrectangle being a leaf of T_c^+ or in a region $R \setminus C$, where (R, C) is a so-called *compressed edge* of T_c^+ [10]. As the number of cells and the number of hyperrectangles and regions are bounded by $\mathcal{O}(\sqrt{|S|})$, this correspondence can be obtained by a brute-force approach performing $\mathcal{O}(\text{scan}(|S|))$ memory transfers. To compute the desired correspondence between the points in S and the hyperrectangles and regions, we first sort both lists lexicographically according to the cell entries. By scanning both sorted lists, we can subsequently augment each point with a representation of the hyperrectangle rather or region which contains the point. Both steps can be performed in $\mathcal{O}(\text{sort}(|S|))$ memory transfers.

Finally, in the third phase, Govindarajan *et al.* replace every compressed edge by a sequence of splits which results in the desired tree T''. They prove that the resulting tree T'' has size $\mathcal{O}(|S|)$. As all techniques in this step are only based on scanning and sorting, the expansion of all edges can be performed cache-obliviously using $\mathcal{O}(\text{sort}(|S|))$ memory transfers.

Step 3: Construction of $\mathbf{T'}$ Given the pseudo split tree T'' of S, Govindarajan *et al.* apply time-forward processing to remove every box R with $R \cap S = \emptyset$ from T'' and to compress all paths consisting of nodes having one child to a single edge in the resulting tree. The overall process is only based on scanning, sorting and time-forward processing and can therefore be implemented efficiently in the cache-oblivious model using $\mathcal{O}(\text{sort}(|S|))$ memory transfers. The obtained tree is the desired fair split tree of S. As the bound for the space consumption follows from the layout of the algorithm, this completes the proof of the following lemma and, hence, also the proof of Lemma 1:

Lemma 2 *Given a set S of points in $\mathbb{R}^d$ and a constant $\alpha \in [1 - \frac{1}{4d}, 1)$, function* PartialFairSplitTree *(Algorithm 2) computes a partial fair split tree T' of S in $\mathcal{O}(\text{sort}(|S|))$ memory transfers using $\mathcal{O}(|S|/B)$ blocks of external memory. Each leaf of T' represents a subset of S with at most $|S|^\alpha$ points.*

2.3 Construction of the Well-Separated Pairs

Once a fair split tree for the point set S is computed, Govindarajan *et al.* simulate the internal memory algorithm for constructing the pairs $\{A_i, B_i\}$ of the designated WSPD in external memory by applying the time-forward processing technique performing an overall number of $\mathcal{O}(\text{sort}(|S|))$ I/Os. Besides time-forward processing, the I/O-efficient algorithm is only based on scanning and sorting. Thus, the complete procedure can be implemented in a cache-oblivious manner in $\mathcal{O}(\text{sort}(|S|))$ memory transfers as well.

Theorem 3 *Given a set S of points in $\mathbb{R}^d$ and a real constant $s > 0$, a well-separated pair decomposition for S with separation ratio s having size $\mathcal{O}(s^d|S|)$ can be computed in $\mathcal{O}(\text{sort}(|S|))$ memory transfers using linear space.*

3 Applications

Constructing t-spanners with a linear number of edges can be performed easily in a cache-oblivious manner using the WSPD. Given a set S of points in $\mathbb{R}^d$ and a real constant $t > 1$, construct a WSPD for S with separation ratio $4\frac{t+1}{t-1}$ having size $\mathcal{O}(|S|)$ and add, for each pair $\{A_i, B_i\}$ of the WSPD, exactly one (arbitrary) pair $\{x, y\}$ of points with $x \in A_i$ and $y \in B_i$ to an initially empty edge set E. Callahan and Kosaraju [6] proved that the resulting graph $G = (S, E)$ is a t-spanner for S with a linear number of edges.

Corollary 4 *Given a point set $S \subset \mathbb{R}^d$ and some constant $t > 1$, we can compute a t-spanner $G = (S, E)$ of linear size using $\mathcal{O}(\text{sort}(|S|))$ memory transfers.*

Using Theorem 3, it is straightforward to restate [10, Theorem 4] in a cache-oblivious version:

Corollary 5 *Given a point set $S \subset \mathbb{R}^d$ and some constant $t > 1$, we can compute a t-spanner $G = (S, E)$ of linear size and diameter at most $2\log|S|$ using $\mathcal{O}(\text{sort}(|S|))$ memory transfers.*

In combination with our previous results for I/O-efficiently pruning dense spanners [11] we can prove:

Lemma 6 ([9, Theorem 6.4.8]) *Given a geometric graph $G = (S, E)$ which is a t-spanner for S for some constant $t > 1$ and given a constant $\varepsilon > 0$, we can compute a $(1+\varepsilon)$-spanner $G' = (S, E')$ of G with $E' \subseteq E$ and $|E'| \in \mathcal{O}(|S|)$ using $\mathcal{O}(\text{sort}(|E|))$ memory transfers.*

Acknowledgments

We thank Norbert Zeh for helpful discussions regarding the I/O-efficient WSPD algorithm [10].

References

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

[2] L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.

[3] L. Arge, M. A. Bender, E. D. Demaine, B. Holland-Minkley, and J. I. Munro. An optimal cache-oblivious priority queue and its application to graph algorithms. *SIAM J.Computing*, 36(6):1672–1695, 2007.

[4] G. S. Brodal and R. Fagerberg. Funnel heap - a cache oblivious priority queue. In *Proc. 13th Intl. Symp. Algorithms and Computation*, volume 2518 of *LNCS*, pages 219–228. Springer, 2002.

[5] P. B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications.* Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1995.

[6] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th Symp. Discrete Algorithms*, pages 291–300, 1993.

[7] 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.

[8] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Symp. Foundations of Computer Science*, pages 285–299, 1999.

[9] F. Gieseke. Algorithmen zur Konstruktion und Ausdünnung von Spanner-Graphen im Cache-Oblivious-Modell. Technical Report TR07-3-005, Universität Dortmund, Fachbereich Informatik, July 2007. In German.

[10] S. Govindarajan, T. Lukovszki, A. Maheshwari, and N. Zeh. I/O-efficient well-separated pair decomposition and its applications. *Algorithmica*, 45(4):585–614, 2006.

[11] J. Gudmundsson and J. Vahrenhold. I/O-efficiently pruning dense spanners. In *Revised Selected Papers Japanese Conf. Discrete and Computational Geometry*, volume 3742 of *LNCS*, pages 106–116. Springer, 2005.

[12] N. Zeh. *I/O-Efficient Algorithms for Shortest Path Related Problems.* PhD thesis, School of Computer Science, Carleton University, 2002.

Computer Aided Design System for Escher-Like Tilings

Hiroshi Koizumi [*] Kokichi Sugihara [†]

Abstract

This paper proposes an efficient method for "Escherization", that is, for generating a tile which is close to a given shape and whose copies cover the plane without gap or overlap except at their boundaries. In this method the "Escherization" is reduced to finding the maximal eigenvalue of a matrix. Hence it is very efficient when compared with the previous method, in which simulated annealing is used. This method is implemented as a computer-aided system, in which a user gives an arbitrary non-intersecting closed curve, and the system automatically finds a placement rule and an associated tile whose boundary is the closest to the given curve.

1 Introduction

A tiling of the plane is a collection of figures, called tiles, that cover the plane without gaps or overlaps except at their boundaries. Tilings have been used for various purposes such as ornament for construction and design of clothes. The patterns made by tilings range from those made by simple geometrical figures to those made by intricate figures, and they attract a great deal of interest of both artists and mathematicians. From a mathematical point of view, the placement rules, incidence types, and other properties of tilings have been deeply considered [1].

Dutch woodblock artist M. C. Escher is one of the greatest artists who created artistic tilings. His works include many artistic tilings made of animal form tiles based on his own try-and-error approaches.

Kaplan and Salesin [3] first introduced the problem of finding tilings automatically whose tiles are similar to a given goal shape. This problem is called the Escherization problem named after Escher and his elegant work. More precisely the Escherization problem is as follows:

> **Escherization problem** Given a closed plane figure S (the "goal shape"), find a closed figure T such that:
>
> 1. T is as close as possible to S; and
> 2. copies of T fit together to form a tiling of the plane.

Kaplan and Salesin proposed a solution to this problem using simulated annealing and concluded that their system performs well on convex or nearly convex shapes but it does not work well on intricate shapes. In addition, it seems that their system requires much time in computation because of the property of simulated annealing. Kaplan and Salesin also published the paper about Dihedral Escherization that deal with a tiling made from two tiles [2].

In this paper, we consider the Escherization problem with a different approach. By reformulating the Escherization problem as a more tractable optimization problem, we propose a new method that is efficient and is able to deal with intricate figures to some extent.

In Section 2 we describe a mathematical basis of tilings, leading into a description of the isohedral tilings, on which this work is based. Then in Section 3 we introduce a measure of closeness between two figures, formulate the Escherization problem as an optimization problem, and reduce it to an eigenvalue problem. Section 4 presents examples of the behavior of our system. Finally, section 5 discusses further refinements and directions of future work.

2 Basis of Tilings

In this section, we review a mathematical basis of tilings, in particular, of the isohedral tilings.

2.1 Definitions

A *plane tiling* $\mathcal{T}$ is defined as a countable family of closed sets, called tiles, $\mathcal{T} = \{T_1, T_2, \ldots\}$ which fulfills the next two conditions:

$$\begin{aligned} \textstyle\bigcup_i T_i &= \mathbb{R}^2, \\ T_i^o \cap T_j^o &= \emptyset \ (\forall i, j,\ i \neq j), \end{aligned}$$

where T_i^o is the interior of tile T_i and $\mathbb{R}^2$ means the Euclidean plane. Here we additionally require that each tile in a tiling is a topological disk. From the above definition, the intersection of any set of tiles are either empty, points or curves. A point shared by three or more tiles is called a *tiling vertex* and a curve shared by two tiles is called a *tiling edge*.

A *symmetry* of a figure in the plane is an isometry which leaves the figure unchanged. It is easy to see

[*]Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, hiroshi_koizumi@mist.i.u-tokyo.ac.jp

[†]Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, sugihara@mist.i.u-tokyo.ac.jp

that the set of symmetries of a figure has a natural group structure under composition of isometries as the operation. Therefore, the set of symmetries of a figure is called the symmetry group of that figure.

Let $S(\mathcal{T})$ be the symmetry group of the tiling $\mathcal{T}$. Two tiles T_1, T_2 of a tiling $\mathcal{T}$ are said to be *transitively equivalent* if $T_2 = sT_1$ for some $s \in S(\mathcal{T})$. Transitive equivalence is an equivalence relation and partitions the tiles into equivalence classes. The collection of all tiles of $\mathcal{T}$ that are transitively equivalent to T_1 is called the *transitivity class* of T_1. If all tiles of $\mathcal{T}$ form one transitivity class, we say that $\mathcal{T}$ is *isohedral.*

It is known that isohedral tilings are classified into only 93 types according to the valences of the tiling vertices and the incidence symbols whose definition is given later [1]. Each type is denoted as IH01, IH02, ..., IH93. In addition, the isohedral tilings are so regular that it is easy to deal with them mathematically. At the same time they have enough flexibility to express the tilings made of various shapes of tiles. It is also known that almost all of the Escher's tilings are isohedral [3]. Thus in this paper, we concentrate on the isohedral tilings as a mathematical basis for solving the Escherization problem as Kaplan and Salesin did.

2.2 Properties of the isohedral tilings

In isohedral tiling, the shapes of tiling edges are subject to certain constraints. These constraints are summarized by incidence symbols. Incidence symbols are labeled arrows which are constructed as follows.

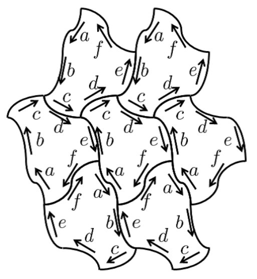


Figure 1: incidence symbols

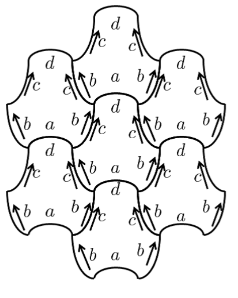


Figure 2: incidence symbols for self-symmetric tiles

Given a tiling, first we take a tile in the tiling and assign a labeled arrow a to an arbitrary chosen tiling edge of the tile. We then proceed along the boundary of this tile in the direction of the arrow and assign labeled arrows $b, c, \ldots$ to tiling edges one by one (see Figure 1). If the symmetry group of the tiling contains an element that maps a tile onto itself as shown in the Figure 2, we assign the same labeled arrows to the associated pair of tiling edges. If an edge is assigned double headed arrow we assume it is unoriented. Next, we copy the labeling of the tile to all the other tiles in the tiling according to the placement rule defined by the symmetry group.

Here, we consider the allowed deformation of the shape of a tile in a isohedral tiling. We can not move all tiling vertices and tiling edges independently. The constraints are obtained from the incidence symbols. See [3] for details.

3 Formulation of the Escherization Problem

In this section we formulate the Escherization problem as a new optimization problem. First we introduce the shape metric which is used as the criterion as to how much two shapes are alike and, we use the metric as objective function of our optimization problem. Next we consider the constraints that should be satisfied by the tiles. Finally we optimize the objective function under the constraints. Then we get the optimal figure that fulfills the two conditions of the Escherization problem.

3.1 The shape metric

Suppose that a figure is represented by a clockwise sequence of points on the boundary. That is, a figure is represented as the $2 \times n$ matrix U, whose i-th column is the coordinates of the i-th vertex. Then we use the metric proposed by Werman and Weinshall [4] for comparing two shapes.

Given two figures, we first transform them so that the centroids are at the origin. Let the resulting figures are represented by the matrices U and W. We define the metric D_{sim} as

$$\begin{aligned} D^2_{\mathrm{sim}}(U, W) &= \min_{s,\mu} \left|\left| sR(\mu)\frac{U}{||U||} - \frac{W}{||W||} \right|\right|^2 \\ &= 1 - \frac{||UW^\top||^2 + 2\det(UW^\top)}{||U||^2||W||^2}, \end{aligned} \quad (1)$$

where $||X||$ is Frobenius norm of the matrix X, s is a scalar and $R(\mu)$ is the matrix of rotation of μ radian. This metric is invariant under rotation, scaling and translation.

3.2 Formulation as an optimization problem

Here, we consider solving the Escherization problem as an optimization problem. To accomplish this, we find a figure that is the closest to a given figure in the metric D_{sim} under the condition that the figure can tile the plane. We formulate this in the following.

Suppose that the goal shape is represented by the matrix W, and the optimal tile which we want to find is represented by the matrix U, where their centroids are already adjusted so that they coincide with the origin of coordinate system. Hence, the elements of U are variables whereas the elements of W are constants. To get the figure U close to the goal shape W, we should minimize the metric $D_{\mathrm{sim}}(U, W)$. From the form of the equation (1), to minimize the distance $D_{\mathrm{sim}}(U, W)$ is equivalent to maximize

$$\frac{||UW^\top||^2 + 2\det(UW^\top)}{||U||^2}. \tag{2}$$

Let $\boldsymbol{u}_x, \boldsymbol{u}_y, \boldsymbol{w}_x, \boldsymbol{w}_y$ be n-dimensional vectors. We represent U, W and $\boldsymbol{u}$ as

$$U = \begin{pmatrix} \boldsymbol{u}_x{}^\top \\ \boldsymbol{u}_y{}^\top \end{pmatrix},\ W = \begin{pmatrix} \boldsymbol{w}_x{}^\top \\ \boldsymbol{w}_y{}^\top \end{pmatrix},\ \boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}_x \\ \boldsymbol{u}_y \end{pmatrix}.$$

Moreover, we represent $UW^\top$ as

$$\begin{aligned} UW^\top &= \begin{pmatrix} \boldsymbol{u}_x{}^\top \\ \boldsymbol{u}_y{}^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{w}_x & \boldsymbol{w}_y \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{u}_x{}^\top \boldsymbol{w}_x & \boldsymbol{u}_x{}^\top \boldsymbol{w}_y \\ \boldsymbol{u}_y{}^\top \boldsymbol{w}_x & \boldsymbol{u}_y{}^\top \boldsymbol{w}_y \end{pmatrix} \\ &= \begin{pmatrix} a & b \\ c & d \end{pmatrix}. \end{aligned}$$

Then, we get

$$\begin{aligned} & ||U||^2 = \boldsymbol{u}_x{}^\top \boldsymbol{u}_x + \boldsymbol{u}_y{}^\top \boldsymbol{u}_y = \boldsymbol{u}^\top \boldsymbol{u}, \\ & ||UW^\top||^2 + 2\det(UW^\top) \\ = {} & (a^2 + b^2 + c^2 + d^2) + 2(ad - bc) \\ = {} & (a + d)^2 + (b - c)^2 \\ = {} & (\boldsymbol{u}_x{}^\top \boldsymbol{w}_x + \boldsymbol{u}_y{}^\top \boldsymbol{w}_y)^2 + (\boldsymbol{u}_x{}^\top \boldsymbol{w}_y - \boldsymbol{u}_y{}^\top \boldsymbol{w}_x)^2 \\ = {} & \left(\boldsymbol{u}^\top \begin{pmatrix} \boldsymbol{w}_x \\ \boldsymbol{w}_y \end{pmatrix}\right)^2 + \left(\boldsymbol{u}^\top \begin{pmatrix} \boldsymbol{w}_y \\ -\boldsymbol{w}_x \end{pmatrix}\right)^2 \\ = {} & \boldsymbol{u}^\top \begin{pmatrix} \boldsymbol{w}_x\boldsymbol{w}_x{}^\top + \boldsymbol{w}_y\boldsymbol{w}_y{}^\top & \boldsymbol{w}_x\boldsymbol{w}_y{}^\top - \boldsymbol{w}_y\boldsymbol{w}_x{}^\top \\ \boldsymbol{w}_y\boldsymbol{w}_x{}^\top - \boldsymbol{w}_x\boldsymbol{w}_y{}^\top & \boldsymbol{w}_x\boldsymbol{w}_x{}^\top + \boldsymbol{w}_y\boldsymbol{w}_y{}^\top \end{pmatrix} \boldsymbol{u}. \end{aligned} \tag{3}$$

Then, the expression (2) turns out to be

$$\frac{\boldsymbol{u}^\top V \boldsymbol{u}}{\boldsymbol{u}^\top \boldsymbol{u}}, \tag{4}$$

where V is

$$V = \begin{pmatrix} \boldsymbol{w}_x\boldsymbol{w}_x{}^\top + \boldsymbol{w}_y\boldsymbol{w}_y{}^\top & \boldsymbol{w}_x\boldsymbol{w}_y{}^\top - \boldsymbol{w}_y\boldsymbol{w}_x{}^\top \\ \boldsymbol{w}_y\boldsymbol{w}_x{}^\top - \boldsymbol{w}_x\boldsymbol{w}_y{}^\top & \boldsymbol{w}_x\boldsymbol{w}_x{}^\top + \boldsymbol{w}_y\boldsymbol{w}_y{}^\top \end{pmatrix}.$$

Note that V is a symmetrical matrix. In addition, from the form of equation (3), V is nonnegative definite.

Next we consider constraint conditions. They vary with types of isohedral tilings. Here we explain the case of IH07 (the tiling type associated with the symmetry group generated by the rotation at the origin by 120 degrees and a translation in an arbitrarily fixed direction and distance) as an example. The basic tile and the associated incidence symbols of this type are as shown in Figure 3.

We place n points on the boundary of a tile as shown in Figure 4. That is, the figure is represented as $U = (P_N, P_{N-1}, \ldots, R'_{N-1})$, where, though each

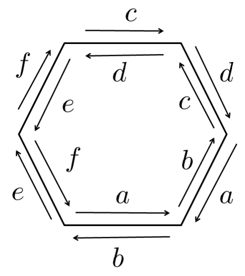


Figure 3: Incidence symbol of IH07

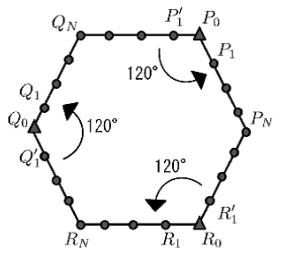


Figure 4: Points on the boundary (IH07)

P_i, Q_i, R_i represents a point, we equally deal with it as a column vector whose elements are its coordinates. In this case, n is equal to $6N$.

The symmetry group of IH07 is generated by the rotation of 120 degrees and a translation. From constraints of the incidence symbol (See Figure 3), each pair of tiling edges whose included angle is $\angle P_0$ or $\angle Q_0$ or $\angle R_0$ must coincide with each other under the rotation of 120 degrees around P_0, Q_0, or R_0, respectively. Then we get the conditions of tiling edges

$$\left\{ \begin{array}{rcll} S(P'_i - P_0) & = & P_i - P_0 & (i = 1, \ldots, N-1), \\ S(Q'_i - Q_0) & = & Q_i - Q_0 & (i = 1, \ldots, N-1), \\ S(R'_i - R_0) & = & R_i - R_0 & (i = 1, \ldots, N-1), \end{array} \right.$$

where S is the matrix which represents the rotation of 120 degrees. We also write down the conditions of tiling vertices in a similar way:

$$\left\{ \begin{array}{rcl} S(Q_N - P_0) & = & P_N - P_0, \\ S(R_N - Q_0) & = & Q_N - Q_0, \\ S(P_N - R_0) & = & R_N - R_0. \end{array} \right.$$

Moreover, the conditions that the centroid of the tile has to coincide with the origin of the plane can be written as

$$\left\{ \begin{array}{rcl} \boldsymbol{u}_x{}^\top \mathbf{1} & = & 0, \\ \boldsymbol{u}_y{}^\top \mathbf{1} & = & 0, \end{array} \right.$$

where $\mathbf{1}$ is the n-dimensional vector whose elements are all 1's.

The above conditions contain no constant terms and are expressed as linear combination of variables. By using a matrix A, the conditions can be written as

$$A\boldsymbol{u} = \mathbf{0}. \tag{5}$$

Considering in a similar way, we can get the constraint conditions for other types of isohedral tilings as the form of equation (5). We only have to consider the constraints for other transformations such as translation or glide reflection.

Summarizing the above observation, we can represent the Escherization problem as the following optimization problem:

$$\begin{aligned} \max \ & \frac{\boldsymbol{u}^\top V \boldsymbol{u}}{\boldsymbol{u}^\top \boldsymbol{u}}, \\ \text{s.t.} \ & A\boldsymbol{u} = \boldsymbol{0}. \end{aligned} \tag{6}$$

Let B be the matrix whose columns are the orthonormal basis of Ker$A$. Then, the equation (5) is equivalent to $\boldsymbol{u} = B\boldsymbol{\xi}$. Therefore, the optimization problem (6) is reduced to the next optimization problem:

$$\max \ \frac{\boldsymbol{\xi}^\top B^\top V B \boldsymbol{\xi}}{\boldsymbol{\xi}^\top \boldsymbol{\xi}}.$$

This problem is equivalent to the problem of finding the maximum eigenvalue of $B^\top VB$ and is solved efficiently.

4 Examples

We implemented our method as a computer-aided system for Escherization. Here we show examples of the behavior of our system. Figure 5 shows the Escherization of a "lizard". Panel (a) shows the pair of the input and output; the larger figure is the goal shape representing a lizard, and the smaller shape is the tile obtained as the optimized shape. Panel (b) is the result of tiling generated by this tile. In this example, the system chose the tiling type associated by the symmetry group generated by the rotation by 90 degrees and a translation.

Another example is shown in Figure 6, where panel (a) represents the goal shape representing an octopus (the larger figure) and the tile obtained by the optimization (the smaller figure), and panel (b) shows the resulting tiling. The chosen tiling type is the same as that in Figure 5.

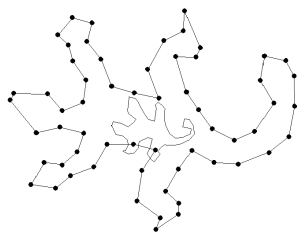
These examples show that our system can deal with complicated figures such as animals with many hands and legs.

In our current system, 10 types of tiling patterns are implemented. Hence for each input figure, the system solves 10 optimization problems and chooses best one according to the shape metric (1). The total time for the computation is about 30s by an ordinary notebook PC with 2.0 GHz cpu and 2 GB RAM.

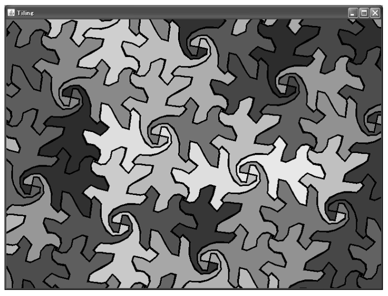
5 Conclusion and future work

In this paper, we reformulate the Escherization problem as an optimization problem being different from the previous work [3]. This approach works well on even somewhat intricate shapes. Additionally, because we don't have to anneal to find the optimal solution, our method is very efficient.

So far, we manually input the vertices of the polygon corresponding to the goal shape. One of our next goals is automatic sampling of vertices from a line drawing of the goal shape. There are many degrees of freedom on how to take the vertices from the line drawing. Thus we have to consider optimal sampling from the viewpoint of the Escherization.

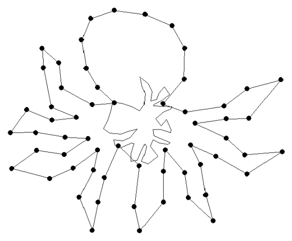


(a)

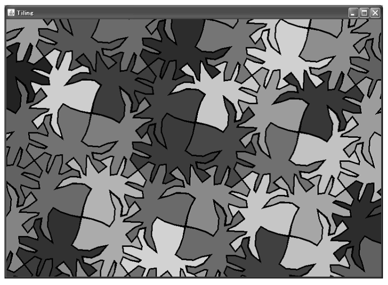


(b)

Figure 5: Escherization of a lizard



(a)



(b)

Figure 6: Escherization of an octopus

Acknowledgments

We would like to express our sincere appreciation to Dr. Imahori for his valuable discussions. This work is supported by the Grant-in-Aid for Scientific Research (B) (No. 20360044) and for Exploratory Research (No. 19650003) of the Japanese Society for Promotion of Science.

References

[1] B. Grunbaum and G. C. Shephard. *Tilings and Patterns.* W. H. Freeman, New York, 1987.

[2] C. S. Kaplan and D. H. Salesin. Dihedral Escherization. *Proceedings of Graphics Interface,* London, May 2004, pp. 255-262.

[3] C. S. Kaplan and D. H. Salesin. Escherization. *Proceedings of SIGGRAPH,* New Orleans, July 2000, pp. 499-510.

[4] M. Werman and D. Weinshall. Similarity and Affine Invariant Distances Between 2D Points Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 17 (1995), pp. 810-814.

Index