ΑΕΙ Ο ΘΕΟΣ Ο ΜΕΓΑΣ ΓΕΩΜΕΤΡΕΙ ΤΟ ΚΥΚΛΟΥ ΜΗΚΟΣ ΙΝΑ ΟΡΙΣΗ ΔΙΑΜΕΤΡΩ ΠΑΡΗΓΑΓΕΝ ΑΡΙΘΜΟΝ ΑΠΕΡΑΝΤΟΝ ΚΑΙ ΟΝ ΦΕΥ ΟΥΔΕΠΟΤΕ ΟΛΟΝ ΘΝΗΤΟΙ ΘΑ ΕΥΡΩΣΙ *

NATIONAL & KAPODISTRIAN UNIVERSITY OF ATHENS * UNIVERSITY OF IOANNINA * UNIVERSITY OF CRETE *

# 22nd European Workshop on Computational Geometry

**March 27-29, 2006**

**European Cultural Center of Delphi**

**Delphi, Greece**

**Abstracts**
**Twenty-second European Workshop on Computational Geometry**
**Delphi, Greece**
**March 27–29, 2006**

**Sponsors**

ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
MINISTRY OF NATIONAL EDUCATION AND RELIGIOUS AFFAIRS

ΕΘΝΙΚΟΝ & ΚΑΠΟΔΙΣΤΡΙΑΚΟΝ ΠΑΝΕΠΙΣΤΗΜΙΟΝ ΑΘΗΝΩΝ
NATIONAL & KAPODISTRIAN UNIVERSITY OF ATHENS

UNIVERSITY OF THE AEGEAN
www.aegean.gr
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

UNIVERSITY OF IOANNINA
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

FORTH
IACM

UNIVERSITY OF CRETE

Commercial Association of Atalanti “Η ΕΝΩΣΙΣ”

# Preface

This volume contains extended abstracts of the papers presented at the *Twenty-second European Workshop on Computational Geometry* held at Delphi, Greece, on March 27–29, 2006. These papers are also available electronically at `http://www.di.uoa.gr/~ewcg06` and at the central EuroCG web site `http://www.eurocg.org`. The workshop has been organized under the auspices of the National and Kapodistrian University of Athens, the University of Crete, the University of Ioannina, and the European Association for Theoretical Computer Science (EATCS).

The program and organizing committees would like to thank all the authors who submitted abstracts and all those who presented their work at the workshop. Especially, we would like to thank Bernard Chazelle, Raimund Seidel, and Monique Teillaud for kindly giving invited lectures. Thanks are also due to the staff of the Cultural Center of Delphi for their assistance with local arrangements in the workshop venue. Last but not least, special thanks go to Bettina Speckmann for providing LaTeX classes used to create these proceedings.

We are grateful to the workshop sponsors for providing financial and technical support: the Hellenic Ministry of National Education and Religious Affairs, the National and Kapodistrian University of Athens, the University of Ioannina, the University of Crete, the University of the Aegean, the Institute of Applied and Computational Mathematics of the Foundation for Research and Technology - Hellas, and the Commercial Association of Atalanti “Η ΕΝΩΣΙΣ”.

Ioannis Emiris
Menelaos Karavelas
Leonidas Palios

## Program Committee

| | |
|---|---|
| Ioannis Emiris | National University of Athens |
| Menelaos Karavelas | University of Crete |
| Leonidas Palios | University of Ioannina |

## Organizing Committee

| | |
|---|---|
| Ioannis Emiris | National University of Athens |
| Menelaos Karavelas | University of Crete |
| Christos Konaxis | National University of Athens |
| Leonidas Palios | University of Ioannina |
| George Tzoumas | National University of Athens |

# Table of Contents

# Where to build a temple, and where to dig to find one

Greg Aloupis* ‖ Jean Cardinal† ‖ Sébastien Collette‡ ‖ John Iacono§ Stefan Langerman¶ ‖

## Abstract

In this paper, we analyze the time complexity of finding regular polygons in a set of $n$ points. We use two different approaches to find regular polygons, depending on their number of edges. Those with $o(n^{0.068})$ edges are found by sweeping a line through the set of points, while the larger polygons are found by random sampling. We can find all the polygons with high probability in $O(n^{2.068+\epsilon})$ expected time for every positive $\epsilon$. This compares well to the $O(n^{2.136+\epsilon})$ deterministic algorithm of Brass [1]. Our method can also be used to find incomplete regular polygons, where up to a constant fraction of the vertices are missing.

## 1 Introduction

The focus of this study is on the detection of regular structure in point sets. Our motivation comes from observations that have been published concerning extraordinary symmetries in the placements of ancient towns, temples and other important locations. Specifically, the oracle of Delphi has been measured to be the apex of isosceles triangles with at least seven pairs of ancient Greek cities[1]. The same is true for the oracle at Dodoni, while the small island of Delos is the apex of at least thirteen isosceles triangles. All three of the central locations were considered to be among the most important of places, and in fact Delphi was considered to be the navel of the world. In general, one may find seemingly countless collinearities, reflective symmetries, partial $n$-gons and networks of isosceles triangles when looking at the graph of cities in the ancient world, from the British Isles to the Middle East.

We will not concern ourselves further questioning whether such structures were carefully constructed or instead an expected result on large complete geometric graphs. However, the topic generates other interesting questions. If one chooses a particular location as a temple, it is not difficult to construct cities (at least on paper) so that the temple becomes the center of several symmetries. What about the opposite? Given a set of existing cities, where should one decide to place a temple? Or, to ask differently, where should one look for a hidden temple?

## 2 Previous Work

Given a set of $n$ points, we wish to find the maximum subset which satisfies a specific symmetry or structure.

The algorithm by Brass [1], for finding maximum symmetric subsets in a set of points, is capable of handling reflective lines, translations, rotational symmetries and repeated sets. The time complexity is $O(n^{2.136+\epsilon})$ for every positive $\epsilon$.

Brass noted that another result of his algorithm was to find all regular polygons contained in the set, which is remarkable because there exist sets of $n$ points containing $O(n^2)$ regular polygons.

The bound of Brass was reached by analyzing the maximum number of possible isosceles triangles formed by a set of points in the plane. Pach and Agarwal [2] gave a simple bound of $O(n^{2+1/3})$ for that problem. This was improved informally by Pach and Tardos [3] to $O(n^{2.136+\epsilon})$ for every positive $\epsilon$.

## 3 Model of Computation

We assume that all coordinates and other values are stored in a format that allows constant time equality testing and hashing. As hashing is only used to speed up one dimensional searches, it can be substituted with a comparison-based structure at a logarithmic addition cost which is absorbed into the $\epsilon$. Furthermore, as exact computation methods are typically not used, comparison based structures can be used to allow equality tests to be substituted with proximity tests for suitably small proximities to compensate for any small discrepancies in the computation.

## 4 Results

We improve the time complexity for detecting maximal regular polygons in point sets, although unlike

*greg.aloupis@ulb.ac.be

†jcardin@ulb.ac.be

‡Aspirant du F.N.R.S., sebastien.collette@ulb.ac.be

§Department of Computer and Information Science, Polytechnic University, 5 MetroTech Center, Brooklyn NY 11201. Research supported in part by NSF grants CCF-0430849 and OISE-0334653. http://john.poly.edu.

¶Chercheur qualifié du F.N.R.S., stefan.langerman@ulb.ac.be

‖Computer Science Department, Université Libre de Bruxelles, CP212, Boulevard du Triomphe, 1050 Bruxelles, Belgium.

[1]For an informal, illustrative and detailed account, see `http://www.geocities.com/sfetel/en/geometry.htm.`

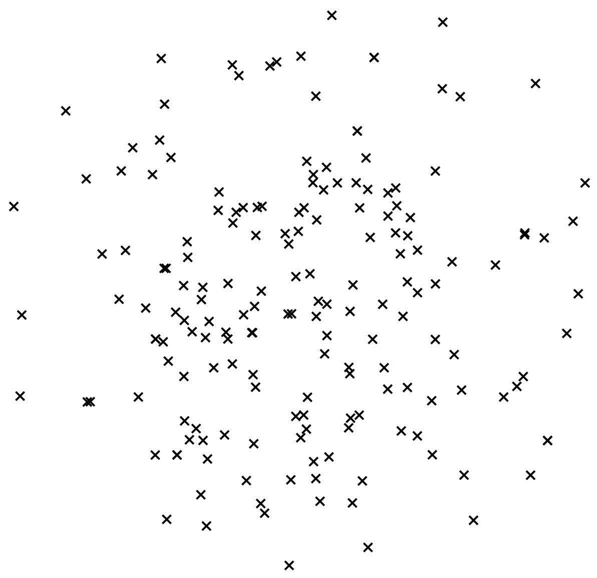


Figure 1: Can you find all of the regular $k$-gons in this figure? Solution is in Figure 2.

the algorithm by Brass, our algorithm is randomized.

Our new bound is $O(n^{2.068+\epsilon})$. Notice that the fractional component in the exponent of $n$ has been halved. This is no coincidence. Our algorithm is designed to reduce this fraction by a factor of 2. Thus, any improvement over the result of Pach and Tardos will be directly reflected in our algorithm.

Our main result appears as Theorem 7 at the end of this section. The remainder of the section is devoted to presenting lemmata that guide us to the Theorem.

**Lemma 1** *In time $O(n^{2.068}\log n)$ we can find all $\leq n^{0.068}$-gons in a set $S$ of $n$ points.*

**Proof.** First, we compute all line segments defined by pairs of points in $S$, and we view this as an embedded graph. We compute a hash table at each vertex containing incident edges which are stored by key value and length.

Let $\phi_i = \pi - \frac{2\pi}{i}$, and $\Phi = \{\phi_3, \phi_4, \ldots \phi_{n^{0.068}}\}$. Thus, $\Phi$ is the set of all angles formed by three adjacent vertices in a regular $\leq n^{0.068}$-gon. Thus, for any edge, it is possible to determine using the set $\Phi$ and the hash tables at its incident vertices if there is a possible neighboring edge that could be in a $\leq n^{0.068}$-gon. This can be done in $O(n^{0.068})$ table lookups, and therefore time.

The algorithm is a line sweep. As we sweep we keep the messages of the following form on the edges that intersect the sweep line: "possible $k$-gon above/below." Edges could carry several messages at one time, and edges deliver their messages to the right endpoint of the edge when the sweep line arrives there.

The sweep proceeds in the normal manner, from left to right, where at vertices we process several types of events:

- Origination Event. This event detects the possible leftmost point of a $k$-gon. If two edges of the same length are to the right of the active vertex and are at angle $\phi_k$, then we give a "possible $k$-gon below" signal to the upper edge and a "possible $k$-gon above signal to the lower edge.

- Propagation Event. This event traces a possible $k$-gon though one edge on its upper or lower portion. If a "possible $k$-gon above/below" signal is received from an edge, and if there is a right facing edge of the same length and angle $\phi_k$ away from the edge delivering the message, the message is propagated to this right facing edge. (The orientation of the angle is determined by whether the signal is an above or below signal). If there is no such right-facing edge, the message is discarded, as what we thought might be a $k$-gon is not.

- Termination Event. This event occurs when we detect the rightmost vertex of a $k$-gon. If a vertex receives a "possible $k$-gon above" and a "possible $k$-gon" below signals from edges on the left of the same length, angle $\phi_k$ apart, and in the proper orientation, then we have finished the process of discovering a complete $k$-gon. We output the description of the $k$-gon (the center, rotation, and gonality can be easily determined from the information at hand), and we do not propagate the two incoming signals.

Hopefully, the correctness of the method is clear from the above description. We now focus on the runtime. It takes $O(n^2 \log n)$ time to perform a line sweep. (The astute reader will realize that a topological sweep could be used instead at a cost of $O(n^2)$, but as all logs get absorbed in the $\epsilon$, this observation is not critical). As there are only $2n^{0.068}$ possible signals, if all of them appeared on all $O(n^2)$ edges, this would create a total of $O(n^{2.068})$ signals to handle over the entire sweep. Propagation and termination events are done with table lookup and take constant time for each event. Generating all origination events takes $O(n^{0.068})$ table lookups for each incident edge to the right, as all angles in $\Phi$ are searched. Thus, the total runtime is $O(n^{2.068})$ if we allow hashing, and $O(n^{2.068}\log n)$ in the model of computation described above. $\square$

**Definition 1** *A* prime-skip triangle *of a $k$-gon is an isosceles triangle formed by three vertices of the $k$-gon where the two non-apex vertices defining the equal sides of the isosceles triangle are at a prime number of vertices away from each other.*

**Observation 1** *Given a prime-skip triangle of a $k$-gon $G$, $k$ and thus $G$ can be uniquely determined in logarithmic time.*

The simplest way to determine which one it is in constant time is to build a lookup table of size $O(n^2)$. If we do not allow hashing, a logarithmic factor is added as we have to look in a binary search tree. The following follows directly from the prime number theorem:

**Observation 2** *Given a $k$-gon $G$, a random isosceles triangle is prime-skip with probability $\frac{1}{\log n}$.*

**Lemma 2** *The sum of complexities of all $\geq n^{0.068}$-gons is at most $n^{2.068+\epsilon}$.*

**Proof.** Let $\kappa_i$ be the number of $i$-gons in a fixed set $S$ of $n$ points. The sum of the complexities of all $\geq n^{0.068}$-gons is $\sum_{i=n^{0.068}}^{n} i\kappa_i$. Any $k$-gon generates at most $O(k^2)$ isosceles triangles of which at most $O(\frac{k^2}{\log n})$ are prime-skip. Thus there are at most $\sum_{i=n^{0.068}}^{n} \frac{i^2\kappa_i}{\log i}$ prime-skip, and therefore distinct, isosceles triangles. We know from [3] that there are at most $O(n^{2.136+\epsilon})$ distinct isosceles triangles and thus:

$$\sum_{i=n^{0.068}}^{n} \frac{i^2\kappa_i}{\log i} = O(n^{2.136+\epsilon})$$

which since $n^{0.068} \leq i \leq n$ gives

$$\frac{n^{0.068}}{\log n} \sum_{i=n^{0.068}}^{n} i\kappa_i = O(n^{2.136+\epsilon})$$

and dividing and absorbing the log into the $\epsilon$ gives

$$\sum_{i=n^{0.068}}^{n} i\kappa_i = O(n^{2.068+\epsilon})$$

This last equation is exactly the statement of the lemma. □

**Definition 2** *A* more-than-half-full *$k$-gon is a subset of the vertices of a regular $k$-gon containing at least $k/2$ points.*

**Corollary 3** *The sum of complexities of all more-than-half-full $\geq n^{0.068}$-gons in a set $S$ of $n$ points is at most $n^{2.068+\epsilon}$*

**Proof.** This is an easy variant of Lemma 2, as the more-than-half-full condition does not change anything but constants buried in the big-O. □

**Lemma 4** *Given an isosceles triangle $T$, we can determine one of the following:*

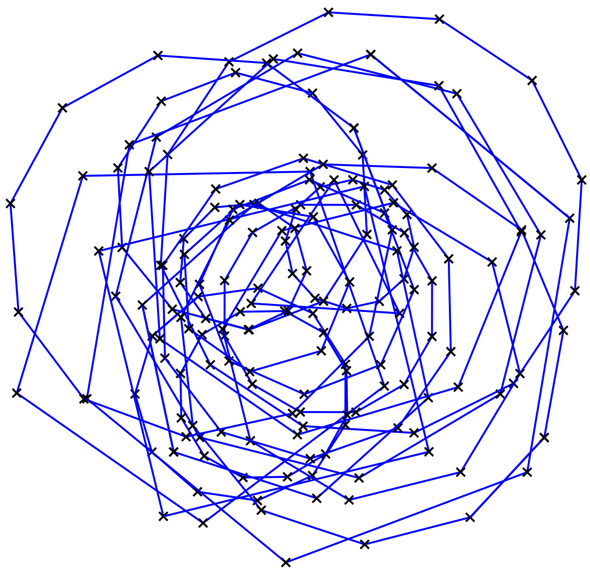


Figure 2: Here we show all of the regular $k$-gons in the point set illustrated in Figure 1.

- *if the candidate $k$-gon containing $T$ is less-than-half-full, then we can answer "$T$ is not part of a $k$-gon" in $O(1)$ time;*
- *otherwise we can decide in $O(k)$ time whether $T$ is part of a $k$-gon or not.*

**Proof.** Given an isosceles triangle $T$, the center of the candidate polygon comes from the circumcenter which can be found in constant time. From the apex angle of the triangle, we can determine what is the smallest value of $k$ such that $T$ could be three vertices of a $k$-gon. This could be done by expressing the angle as a fraction of $2\pi$ in lowest terms, or alternately via a table lookup. Since there are at most $n^2$ possible apex angles in triangles with vertices in $\leq n-$gons, such a table could be precomputed in time $O(n^2)$, which does not asymptotically contribute to the overall runtime.

Given $T$, the center of the polygon, and the value of $k$, one knows the location of all vertices on the candidate polygon. We then check to see if all of the vertices in the candidate polygon are present. We check all $k-3$ of them in random order. If any one vertex is not present we terminate the check and output nothing. If all vertices are present, we output the candidate $k$-gon.

As we always spend at most $O(k)$ time checking each candidate $k$-gon, if the candidate $k$-gon is more-than-half-full, we trivially spend $O(k)$ time. If the candidate $k$-gon is not more-than-half-full, over half of the $k-3$ tests will fail. Since the tests are ordered randomly, we expect to perform at most 2 tests, and thus expect to spend only $O(1)$ time performing the tests in this case. □

**Lemma 5** *For any $\geq n^{0.068}$-gon $G$, at least $\frac{1}{n^2}$ of the isosceles triangles are in $G$.*

**Proof.** According to a personal communication of Pach and Tardos cited in [1], the maximum number of isosceles triangles among $n$ points in the plane is $O(n^{2.136+\epsilon})$. Any $\geq n^{0.068}$-gon $G$ defines $O(n^{0.136})$ isosceles triangles. Thus, $O(1/n^2)$ of the isosceles triangles have all three points in $G$. □

**Corollary 6** *For any $\geq n^{0.068}$-gon $G$, $\frac{1}{n^2 \log n}$ of the isosceles triangles are prime-skip triangles in $G$.*

**Proof.** Follows from Lemma 5 and Observation 2. □

**Theorem 7** *With high probability, we can find all regular polygons in a set $S$ of $n$ points in the plane in expected time $O(n^{2.068+\epsilon})$.*

**Proof.** Using Lemma 1, all regular $\leq n^{0.068}$-gons can be found in time $O(n^{2.068})$. We thus focus on the case of large polygons, that is, $\geq n^{0.068}$-gons.

The algorithm proceeds as follows: we pick $O(n^2 \log^2 n)$ random isosceles triangles formed from the vertices of $S$. Corollary 6 and the solution to the coupon collector problem indicates that we will now have with positive constant probability at least one prime-skip isosceles triangle from every $\geq n^{0.068}$-gon. Lemma 4 explains how we can determine which isosceles triangles come from $k$-gons and which do not. Since $O(1)$ expected time is spent checking those that are not part of a $k$-gon, we spend $O(n^2 \log^2 n)$ time checking all such triangles. In order to check those that are part of a $k$-gon, or at least a partially full $k$-gon, we spend time linear in the gonality for each check. Corollary 3 states that the sum of the gonalities of all more-than-half-full $\geq n^{0.068}$-gons is $O(n^{2.068})$ thus giving this same bound on the time to perform this class of checks. As we picked $O(n^2 \log^2 n)$ triangles, the total complexity is $O(n^{2.068+\epsilon})$.

□

## 5 Variants

While we have focused on the case of finding regular $k$-gons, our algorithm can be used to find all $k$-gons where a constant fraction of the points in the $k$-gon are missing, by using a Monte Carlo version of Lemma 4. The identification of such almost-complete symmetric sets of historical sites can aid the computational archaeologist identify possible locations for exploratory missions in search of buried ruins. This variant has the same runtime as the original algorithm.

Also, by processing all $k$-gons with the same center, more complex types of symmetry can be easily detected.

## 6 Acknowledgments

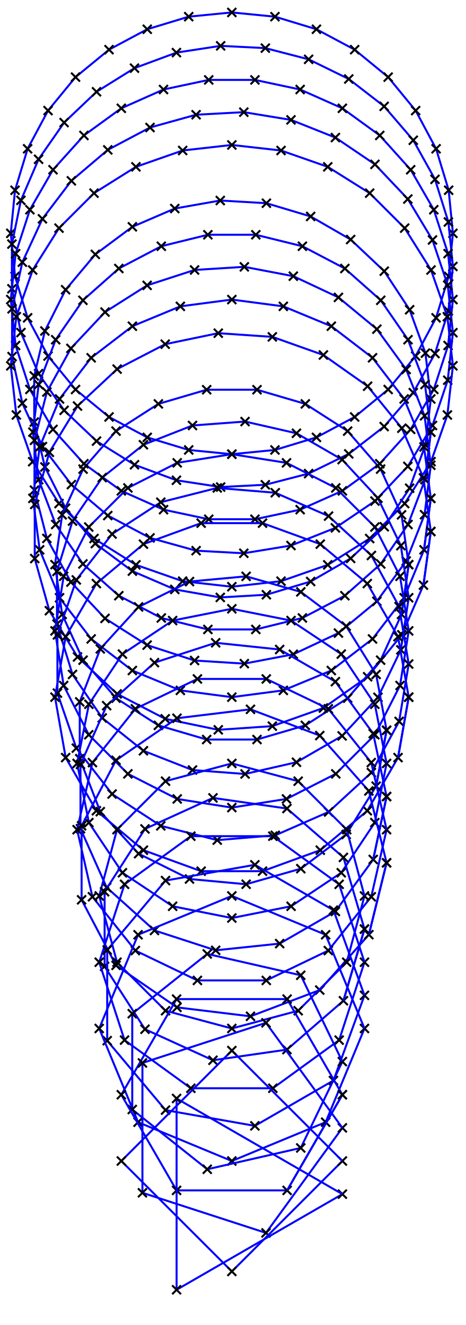
We would like to thank Boris Aronov, Jean Chapelle and Erik Demaine for interesting discussions about chords, polygons, and Euclid in general.

## References

[1] P. Brass. On finding maximum-cardinality symmetric subsets. *Computational Geometry - Theory and Applications*, 24(1):19–25, 2003.

[2] J. Pach and P. K. Agarwal. *Combinatorial geometry*. Wiley-Interscience, 1995.

[3] J. Pach and G. Tardos. Personal communication cited in [1].



# Fixed Parameter Algorithms for Minimum Weight Partitions

Christian Borgelt[†], Magdalene Grantson[*], and Christos Levcopoulos[*]

## Abstract

In this paper we propose to solve two hard geometric optimization problems: We describe a fixed parameter algorithm for computing the minimum weight triangulation (MWT) of a simple polygon with $(n-k)$ vertices on the perimeter and $k$ hole vertices in the interior, that is, for a total of $n$ vertices. We show that the MWT can be found in time at most $O(n^4 4^k k)$, and thus in time polynomial in $n$ if $k \leq O(\log n)$. We implemented our algorithm in Java and report experiments backing our analysis.

Given a convex polygon with $(n-k)$ vertices on the perimeter and $k$ hole vertices in the interior, that is, for a total of $n$ vertices, we also describe a fixed parameter algorithm for computing the minimum weight convex partition (MWCP) of the input. We show that the MWCP problem can be found in at most $O(n^3 \cdot k^{4k-8} \cdot 2^{13k})$ time, and thus at $O(n^3)$ time if $k$ is constant, and in time polynomial in $n$ if $k = O(\frac{\log n}{\log \log n})$. Our results for the MWCP problem hold also for the more general case where the input is an $n$-vertex PSLG and $k$ is the total number of holes and/or reflex vertices inside the convex hull.

## 1 Introduction

We propose to solve two hard geometric optimization problems in this paper:

1. We consider the problem of finding the *minimum weight triangulation* (MWT) of a simple polygon with $n-k$ vertices on the perimeter and $k$ hole vertices in the interior, i.e., for a total of $n$ vertices. In this case, a MWT is a maximal set of non-intersecting edges with minimum total edge length, all of which lie inside the polygon. The problem of finding the MWT of a set $S$ of points can be reduced to this problem by finding the convex hull of $S$, which is then treated as a (convex) polygon, while all vertices not on the convex hull are treated as hole vertices. The MWT problem has several applications [17, 15].

2. We also consider the problem of finding the *minimum weight convex partition* (MWCP) of a convex polygon with $n-k$ vertices on the perimeter and $k$ hole vertices in the interior, i.e., for a total of $n$ vertices. In this case, the MWCP is a convex partition such that the total edge length is minimised. The MWCP has applications in computer graphics [17], image processing [15], database systems [13], and data compression [17].

It is known that the MWCP of $G$ is NP-hard [11]. With respect to a convex polygon with a single hole vertex or a convex polygon with two hole vertices, we [7] gave $O(n)$ and $O(n^2)$ time algorithms, respectively. With respect to $n$-vertex polygons without holes, Keil [9] developed a dynamic programming algorithm that runs in $O(n^2 N^2 \log n)$ time, where $N$ is the number of concave vertices. Later Agarwal, Flato, and Halperin [1] improved this time bound to $O(n^2 N^2)$.

The complexity status of the MWT problem has been open since 1975, when it was included in a list of problems neither known to be NP-complete or solvable in polynomial time [5]. Recently, however, it was reported that the MWT problem is NP-hard [16]. For $k = 0$ (no holes), however, a MWT can be found by dynamic programming in time $O(n^3)$ [6, 12].

Recent attempts to give exact algorithms for computing a MWT and a MWCP exploit the idea of developing a so-called *fixed parameter algorithm*. Such an algorithm has a time complexity of $O(n^c \cdot f(k))$, where $n$ is the input size, $k$ is a (constrained) parameter, $c$ is a constant independent of $k$, and $f$ is an arbitrary function [3].

W.r.t. a MWT of a simple polygon with holes the total number $n$ of vertices is the size of the input and we may choose the number $k$ of hole vertices as the constrained parameter. An algorithm based on such an approach was presented in [10] and analyzed to run in $O(n^5 \log(n)\, 6^k)$ time. In this paper we describe a fixed parameter algorithm inspired by a basic observation in this algorithm, but deviating in several respects. Due to improvements in both the algorithm and its analysis, we are able to show that the time needed to find a MWT of a polygon with hole vertices is at most $O(n^4 4^k k)$. In addition, we implemented our algorithm in Java and performed experiments backing our analysis.

W.r.t. a MWCP of a convex polygon with holes the total number $n$ of vertices is the size of the input and we may choose the number $k$ of hole vertices as the

[*]Department of Computer Science, Lund University, Box 118, 221 Lund, Sweden, {magdalene,christos}@cs.lth.se, [†]Department of Knowledge Processing and Language Engineering, University of Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany, borgelt@iws.cs.uni-magdeburg.de

constrained parameter. In this paper, we describe a fixed parameter algorithm for computing the MWCP of a convex polygon $G$ with $n-k$ vertices on the perimeter and $k$ hole vertices in the interior, that is, for a total of $n$ vertices. We show that our algorithm can solve the problem in at most $O(n^3 \cdot k^{4k-8} \cdot 2^{13k})$ time, and thus at $O(n^3)$ time if $k$ is constant, and in time polynomial in $n$ if $k = O(\frac{\log n}{\log\log n})$. Our results for the MWCP problem hold also for the more general case where the input is an $n$-vertex PSLG and $k$ is the total number of holes and/or reflex vertices inside the convex hull.

The paper is structured as follows. In Section 2 we discuss our results for the MWT problem. In section 3 we discuss our results for the MWCP problem.

The full versions of both algorithms can be found in [7] and [8] respectively.

## 2 A Fixed Parameter Algorithm for the MWT Problem

In this Section we will discuss our results for the MWT problem.

### 2.1 Preliminaries and Basic Idea

Following [2], we call a polygon with holes a *pointgon* for short. We denote the set of $(n-k)$ perimeter vertices by $V_p = \{v_1, v_2, \ldots, v_{n-k}\}$, assuming that they are numbered in counterclockwise order starting at an arbitrary vertex. The set of $k$ hole vertices we denote by $V_h = \{v_{n-k+1}, v_{n-k+2}, \ldots, v_n\}$. The set of all vertices is denoted by $V = V_p \cup V_h$, the pointgon formed by them is denoted by $G$.

**Definition 1** *A vertex $u \in V$ is said to be* lexicographically smaller *than a vertex $v \in V$, written $u \prec v$, iff (1) the x-coordinate of $u$ is smaller than the x-coordinate of $v$ or (2) the x-coordinate of $u$ is equal to the x-coordinate of $v$, but the y-coordinate of $u$ is smaller than the y-coordinate of $v$.*

W.l.o.g. we assume that the vertices in $V_h$ are in lexicographical order, i.e., $\forall i; n-k < i < n : v_i \prec v_{i+1}$. (Otherwise we can sort and renumber them.)

**Definition 2** *A path in a pointgon $G$, i.e., a sequence of vertices from $V$, is called* lexi-monotone *iff it is either lexicographically increasing or lexicographically decreasing. A* separating lexi-monotone path *(or simply a* separating path*) is a lexi-monotone path with start and end vertices on the perimeter of $G$ (i.e. vertices in $V_p$) and a (possibly empty) sequence of hole vertices (i.e. vertices in $V_h$) in the middle, which does not intersect the perimeter of $G$.*

With these definitions, the core idea of our algorithm (as well as the core idea of the algorithm in [10]) is based on the following observation:

**Observation 1** *Let $v \in V_p$ be an arbitrary vertex on the perimeter of a pointgon $G$. Then in every triangulation $T$ of $G$ there exists:* either *a separating path $\pi$ starting at $v$* or *two perimeter vertices $v_c$ and $v_{cc}$ that are adjacent to $v$ and that together with $v$ form a triangle without any hole vertices in its interior.*

As a consequence, we can try to find the MWT of a given pointgon $G$ with a recursive procedure that considers possible splits of $G$ into at most two sub-pointgons (using the above observation). The MWT is then obtained as the minimum over all these splits.

Formally, we can describe the solution procedure as follows: Let $G$ be a given pointgon and $v \in V_p$ an arbitrary vertex on the perimeter of $G$. Let $\Pi(G, v)$ be the set of all separating paths of $G$ starting at $v$. If $\pi \in \Pi(G, v)$ is a separating path, let $|\pi|$ be the length of $\pi$ and $L(G, \pi)$ and $R(G, \pi)$ the sub-pointgons to the left and to the right of $\pi$, respectively. Furthermore, let $v_c$ and $v_{cc}$ be the perimeter vertices that are adjacent to $v$ in clockwise and counterclockwise direction, respectively. Then the weight of a MWT of $G$ can be computed recursively as

$$\min \Big\{ \min_{\pi \in \Pi(G,v)} \{ \mathrm{MWT}(L(G,\pi)) + \mathrm{MWT}(R(G,\pi)) - |\pi| \}, \mathrm{MWT}(R(G,(v_{cc},v_c))) + |(v,v_{cc})| + |(v,v_c)| \Big\}.$$

The first term in the outer minimum considers all splits by separating lexi-monotone paths. The second term in the outer minimum refers to the special path $(v_{cc}, v_c)$ that "cuts off" $v$ from the rest of the pointgon if the triangle $(v, v_{cc}, v_c)$ does not contain any hole vertices. Although the above recursive formula only computes the weight of a MWT, it can easily be extended to yield the edges of a MWT by returning the set of edges that is added in order to achieve a triangulation for each recursive call. The union of these sets of edges for the term that yields the minimum weight is a MWT for the original pointgon $G$.

### 2.2 Dynamic Programming

To apply dynamic programming, we have to identify the different subproblems that we meet in the recursion, and we have to find a representation for them. The core idea here is: if in the recursion we prefer to use the same vertex $v$ for attaching separating paths as in the preceding split, every subproblem we encounter can be described by one or two lexi-monotone paths that start at the same vertex $v$ (which we call an *anchor* of the subproblem) and a coherent piece of the perimeter of the input pointgon. An analysis of this statement, providing a proof, will be given later.

We represent a subproblem by an index word over an alphabet with $n$ characters, which uniquely identifies each subproblem. This index word has the general form $(v, \pi_{cc}, \pi_c)$ and describes a counterclockwise

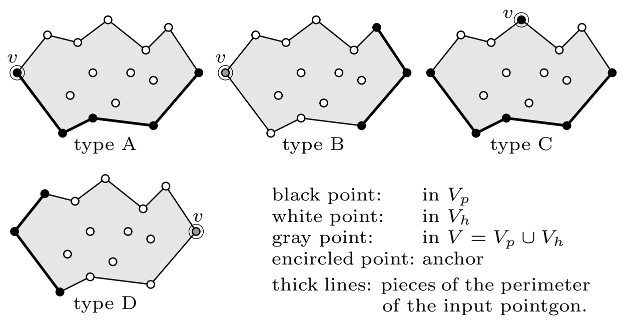


Figure 1: The four types of pointgons we encounter.

walk along the perimeter of the subproblem. The first element is the anchor $v$, which may be a perimeter vertex or a hole vertex of the input pointgon and thus can have $n$ possible values. $\pi_{cc}$ and $\pi_c$ describe the sequences of hole vertices of the input pointgon that are on the separating paths. All elements of $\pi_{cc}$ and $\pi_c$ are in $V_h$—with the possible exception of the last elements, which may be perimeter vertices $s_{cc}$ and $s_c$, respectively. The vertices in a coherent perimeter piece between the end vertices $s_{cc}$ and $s_c$ (if such a perimeter piece exists) are not part of the subproblem representation, but are left implicit.

Instead of pure dynamic programming, we use *memorized tree recursion* based on a trie structure, which is accessed through the index word representing a subproblem. In each recursive call, we first access the trie structure in order to find out whether the solution to the current subproblem is already known. If it is, we simply retrieve and return the solution. Otherwise we carry out the split computations and in the end store the found solution in the trie.

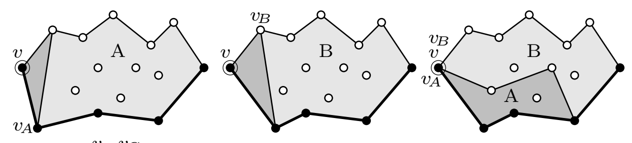
## 2.3 Types of Pointgons

Apart from the input pointgon, which is of neither of these types, we encounter four types of sub-pointgons (see Figure 1; this set differs from the one used in [10]):

**Type A** pointgons have only one separating path starting at the anchor $v$, which must be on the perimeter of the input pointgon. The vertices on the path are lexicographically increasing. There is also a coherent perimeter piece of the input pointgon.

**Type B** pointgons are bounded by two separating paths starting at the anchor $v$, which may be either a perimeter vertex or a hole vertex of the input pointgon. The vertices on both paths are lexicographically increasing. There may or may not be a coherent perimeter piece of the input pointgon.

**Type C** pointgons are bounded by two separating paths starting at the anchor $v$, which must be a perimeter vertex of the input pointgon. One of them



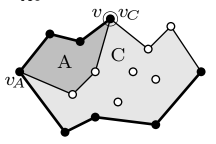


Figure 2: Processing of Type A pointgons in the recursion. $v$ and $v_*$, $* \in \{A, B, C\}$, mark the subproblem anchors.

is lexicographically increasing, the other decreasing. There is a perimeter piece of the input pointgon.

**Type D** pointgons are bounded by two separating paths starting at the anchor $v$, which may be either a perimeter vertex or a hole vertex of the input pointgon. The vertices on both paths are lexicographically decreasing. There must be a perimeter piece of the input pointgon, which contains at least two vertices.

The general principle of the choice of the anchor of a subproblem is that it is the leftmost vertex on the separating path if there is just one path, and the vertex that is on both paths if there are two separating paths. If there are two vertices that are on both paths (because they share both start and end vertex), we choose the leftmost of the two. For the input pointgon, we choose the lexicographically smallest perimeter vertex as the anchor.

For the input pointgon, regardless of whether the path starts at the anchor or cuts off the anchor, we obtain a sub-pointgon of type A for the subproblems. The other types of pointgons can only be created in deeper levels of the recursion. In the following we consider how these types of pointgons are treated in the recursion in our algorithm and thus also prove that these are the only types of pointgons that occur.

**Type A:** The different splits of a type A pointgon are sketched in Figure 2. On the very left a path "cutting off" the anchor, which is seen as leading from the counterclockwise neighbor of $v$ to its clockwise neighbor, can be merged with the existing separating path to give a new type A pointgon. Otherwise, we obtain a type B pointgon (second sketch). For a path starting at the anchor, we distinguish whether it is lexicographically increasing (third sketch) or decreasing (fourth sketch, note the different anchor). In the former case, we obtain one type A and one type B pointgon, which receive the same anchor as the original pointgon. In the latter case, we obtain one type A pointgon, with its anchor at the end of the new separating path, and one type C pointgon, with its anchor equal to that of the original pointgon. Note that all cases may also occur mirrored at a horizontal axis, which should also be kept in mind for the other types.

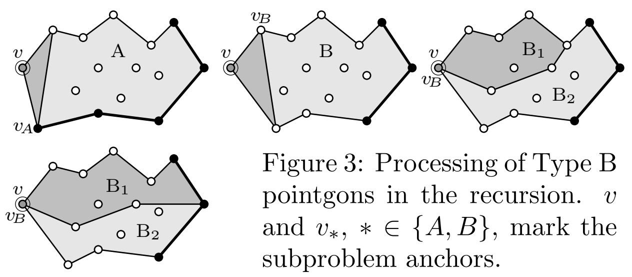


Figure 3: Processing of Type B pointgons in the recursion. $v$ and $v_*$, $* \in \{A, B\}$, mark the subproblem anchors.

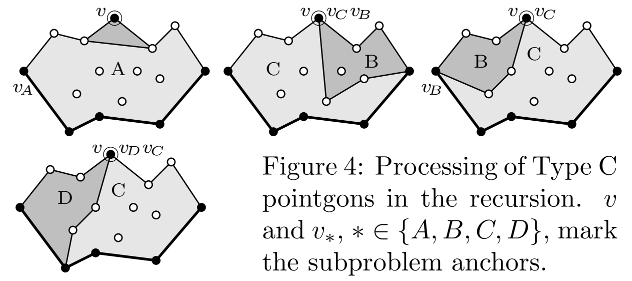


Figure 4: Processing of Type C pointgons in the recursion. $v$ and $v_*$, $* \in \{A, B, C, D\}$, mark the subproblem anchors.

**Type B:** Type B pointgons behave similarly to type A pointgons (see Figure 3). Again we have to check whether a type A pointgon can result (first sketch). Otherwise we get a type B pointgon with an anchor that is one end of the cutting path (second sketch). For separating paths starting at the anchor only type B pointgons can result (third and fourth sketch), since both separating paths are increasing.

**Type C:** Type C pointgons (which do not appear in [10]) are the most complicated case (see Figure 4). If the anchor is “cut off”, we only have one separating path, so the anchor is set to its starting vertex and we obtain a type A pointgon (first sketch). If a separating path is attached to the anchor, we have to distinguish whether it is lexicographically increasing or decreasing. Increasing paths are simpler, leading to a split into one type C and one type B pointgon (second sketch). If the path is lexicographically decreasing, we have to check whether there is a perimeter piece of the input pointgon with at least two vertices. If there is not, we obtain one type B pointgon, with its anchor at its leftmost vertex, and one type C pointgon, which maintains the anchor of the original pointgon (third sketch). Otherwise we obtain one type D and one type C pointgon, both of which receive the anchor of the original pointgon (fourth sketch).

**Type D:** Type D pointgons behave symmetrically to type B pointgons. When the anchor is “cut off” we also have to check whether a type A pointgon results (first sketch). Otherwise we get a type D pointgon with an anchor that is one end of the cutting path (second sketch). For separating paths starting at the anchor either one type B and one type D pointgon (third sketch), namely if one perimeter piece is empty, or two type D pointgons result (fourth sketch).

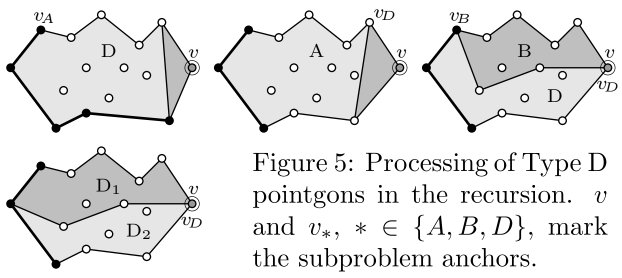


Figure 5: Processing of Type D pointgons in the recursion. $v$ and $v_*$, $* \in \{A, B, D\}$, mark the subproblem anchors.

**Comparison to [10]:** Our approach gives rise to a different set of sub-pointgons. An important advantage of our approach is that it uses a coherent scheme for processing the sub-pointgons, which is strictly based on either “cutting off” the anchor or attaching a lexi-monotone path to the anchor (Section 2.1). In contrast to this, the approach of [10] needs an additional split type (when processing a type 1/type A pointgon).

## 2.4 Analysis

To estimate the time complexity of our algorithm, we group the subproblems and analyze the groups separately. The groups are defined by the number of hole vertices the sub-pointgon has on its perimeter.

So consider the number of subproblems with $l$, $0 \leq l \leq k$, hole vertices on the perimeter. The worst case is that we have three perimeter vertices of the input pointgon, namely the anchor and the two ends of the separating paths. This gives us a factor of $n^3$. Next we have to choose $l$ of the $k$ input hole vertices, for which we have $\binom{k}{l}$ possibilities, and then we have to distribute the chosen hole vertices on the two paths, for which there are $2^l$ possibilities. As a consequence we have in the worst case $O(n^3\binom{k}{l}2^l)$ possible sub-pointgons with $l$ holes on the perimeter.

Given a sub-pointgon with $l$ holes on the perimeter, there are at most $k - l$ holes left to form a separating path and at most $n$ end points. This gives us a maximum of $n2^{k-l}$ possible paths. For each path, we have to check whether it intersects the perimeter of the sub-pointgon. This check can exploit a preprocessing step in which we determine for each edge that could be part of a separating path whether it intersects the perimeter of the input pointgon or not. The resulting table has a size of at most $n^2$. With this table we can check in $O(k - l)$ whether a given separating path intersects a (possibly existing) perimeter piece. We also have to check for an intersection with the at most two already existing separating paths, which contain at most $l + 2$ edges. By exploiting that all paths are lexi-monotone, this check can be carried out in $O(k)$. Once a path is found to be valid, the sub-pointgons have to be constructed by collecting their at most $k + 3$ defining vertices, and their

| $n-k$ | $k$ | time in seconds | time/$n^4 4^k k$ |
|---|---|---|---|
| 3 | 1 | 0.008 ± 0.000 | $7.813\cdot 10^{-6}$ |
| 6 | 1 | 0.009 ± 0.000 | $9.371\cdot 10^{-7}$ |
| 9 | 2 | 0.039 ± 0.003 | $8.324\cdot 10^{-8}$ |
| 12 | 3 | 0.071 ± 0.004 | $7.305\cdot 10^{-9}$ |
| 15 | 4 | 0.121 ± 0.020 | $9.067\cdot 10^{-10}$ |
| 18 | 5 | 0.370 ± 0.090 | $2.582\cdot 10^{-10}$ |
| 21 | 6 | 1.365 ± 0.539 | $1.045\cdot 10^{-10}$ |
| 24 | 7 | 5.402 ± 2.209 | $5.100\cdot 10^{-11}$ |
| 27 | 8 | 25.335 ± 10.449 | $3.220\cdot 10^{-11}$ |
| 30 | 9 | 90.641 ± 34.399 | $1.661\cdot 10^{-11}$ |

Table 1: Results obtained from our Java implementation. All results are averages over 20 runs.

solutions have to be looked up. Both operations take $O(k)$ time. Finally the length of the path has to be computed, which takes $O(k-l)$ time. Thus processing one path takes in all $O(k)$ time.

Therefore the overall time complexity is $O(n^4 4^k k)$.

### 2.5 Implementation

As already mentioned, we implemented our algorithm in Java. Example results for different numbers of holes and perimeter vertices (averages over 20 runs, convex pointgons) are shown in Table 1. The test system was an Intel Pentium 4C@2.6GHz with 1GB of main memory running S.u.S.E. Linux 10.0 and Sun Java 1.5.0_03. To check our estimate of the time complexity, we computed the ratios of the measured execution times to the theoretical values. As can be seen, these ratios are decreasing for increasing values of $n$ and $k$, indicating that the theoretical time complexity is actually a worst case, while average results in practice are considerably better. The source code of our implementation can be downloaded at http://fuzzy.cs.uni-magdeburg.de/~borgelt/pointgon.html.

## 3 A Fixed Parameter Algorithm for the MWCP Problem

In this Section we will discuss our results for the MWCP problem.

### 3.1 Preliminaries

We consider as input a convex polygon with $(n-k)$ vertices on the perimeter and $k$ hole vertices, thus a total of $n$ input vertices. We call such a convex polygon with holes a *convex pointgon* for short. We denote the set of perimeter vertices by $V_p = \{v_0, v_1, \ldots, v_{n-k-1}\}$, assuming that they are numbered in counterclockwise order starting at an arbitrary vertex. The set of hole vertices we denote by $V_h = \{v_{n-k}, v_{n-k+1}, \ldots, v_{n-1}\}$. The set of all vertices is denoted by $V = V_p \cup V_h$, the convex pointgon formed by them is denoted by $G$.

Given a convex pointgon $G$ a *p-edge* is an edge from a hole vertex to any vertex on the perimeter.

**Fact 1** *Given a convex pointgon $G$, apart from edges going between hole vertices, at most 3 p-edges incident to a hole vertex are sufficient to induce a convex partition.*

**Proof.** At most three $p$-edges suffice for each hole vertex because if a hole vertex is incident to four $p$-edges, there must be one that can be removed without introducing a concavity at the hole vertex. Removing it also does not introduce a concavity at the perimeter, because the polygon we consider is convex. □

From the proof in Fact 1, we observe a decisive difference between a $p$-edge and an edge going between hole vertices. That is, removing an edge going between hole vertices so that no concavity is introduced at one end point may very well introduce a concavity at the other end point and thus we cannot remove it.

From Fact 1 we deduce a more special case:

**Fact 2** *For any MWCP of $G$ there are at most three p-edges incident to a hole vertex.*

**Proof.** At most three $p$-edges are needed for each hole vertex because of the same reasoning in the proof of Fact 1. Moreover removing the forth $p$-edge decreases the weight for the case of the MWCP problem. □

Given a convex polygon with $k$ vertex holes, at most 3 $p$-edges incident to each hole vertex are needed to induce a minimum convex partition. Thus a maximum of at most $3k$ $p$-edges can be in the solution. Given at most $3k$ $p$-edges we can also check whether they intersect in constant time since $k$ is constant. A simple and obvious way to solve the problem is to use a brute-force approach of examining all the possible ways of selecting the $3k$ $p$-edges. For each possible $3k$ $p$-edges we consider all possible combinations of non-crossing edges going between hole vertices. There are at most $O(2^3 \cdot 59)^k \cdot k^{-6})$ such combinations [4, 18]. Selecting the best over all combinations solves the problem in $O(n^{3k} \cdot (2^3 \cdot 59)^k \cdot k^{-6})$ time.

Given a convex pointgon $G$, we consider partitioning $G$ into so called *v-pieces*. These $v$-pieces serve the purpose to divide the problem into subproblems such that a subproblem denotes a piece of $G$ containing a single hole vertex. The $v$-pieces alone do not necessarily yield a convex partition. The subproblems representing $v$-pieces are solved and the results are combined. The idea is that for any given convex partition **CP**, the $v$-pieces and **CP** are compatible if and only if for each hole vertex $v_h \in V_h$, the $p$-edges incident to $v_h$ in **CP** lie only in one or several $v$-pieces of $v_h$.

**Definition 3** A $v$-piece $(v_h, v_1, v_2)$ is a part of a given polygon $P$ with holes, where $v_h \in V_h$ is a hole vertex and $v_1 \in V_p$ and $v_2 \in V_p$ are (not necessarily distinct) perimeter vertices (the so-called *vital points* of the $v$-piece). It is the part bounded by the two $p$-edges $(v_h, v_1)$ and $(v_h, v_2)$, the so-called *vital edges*, and the part of the perimeter of $G$ that is traversed by a counterclockwise walk from $v_1$ to $v_2$. If a $v$-piece $(v_h, v_1, v_2)$ contains no other hole vertices, we call it a *legal v-piece* (or just a *v-piece*, for short). If a $v$-piece $(v_h, v_1, v_2)$ contains other hole vertices, we call it an *illegal v-piece*.

Note that the $v$-piece $(v_h, v_1, v_2)$ is not the same as the $v$-piece $(v_h, v_2, v_1)$. That is, the order of the vertices $v_1$ and $v_2$ is important, since the part of the perimeter of $G$ that bounds the $v$-piece is defined by a *counter-clockwise* walk from the first vertex to the second. Therefore $(v_h, v_1, v_2)$ and $(v_h, v_2, v_1)$ are complements of each other w.r.t. the convex pointgon $G$; their union is the whole convex pointgon $G$.

The idea of the algorithm is to partition the perimeter of $G$ into (legal) $v$-pieces, such that each hole vertex $h$ has at most 3 $v$-pieces. Three $v$-pieces suffice, because no more than three $p$-edges per hole vertex are needed to achieve a convex partition (see Fact 1). If there were more than 3 $v$-pieces, at most three can contribute $p$-edges to the solution. Therefore we can remove all $v$-pieces that do not contribute without changing the solution. The partition should be such that the $v$-pieces of a coonvex pointgon $G$ put together contain the entire perimeter of $G$ (but not necessarily the entire polygon $P$). Adjacent $v$-pieces, that is, $v$-pieces that have at least one perimeter vertex in common, should belong to different hole vertices.

The core idea of our algorithm is that in order to find a MWCP it suffices to search all possible partitions into $v$-pieces, where the partition is such that two $v$-pieces associated with the same whole do not share a vital point, but all $v$-pieces together cover the perimeter. To show this, we only have to show that any convex partition has a compatible partition into $v$-pieces, from which it can be derived, so that we do not "miss" any convex partition by searching only partitions into $v$-pieces. One can do this by initially constructing an $v$-piece partition by placing one vital point at the end of each $p$-edge, associating it with the hole vertex the $p$-edge leads to. We turn this into a list, by starting at a perimeter vertex and following the perimeter counterclockwise, numbering vertices and collecting vital points. If several vital points are located at the same perimeter vertex, we order them according to the order in which the corresponding $p$-edges are met on a traversal of the perimeter shrunk by some small $\epsilon$. That is we follow a route inside the perimeter which is at a distance $\epsilon$ from the perimeter. Next we remove from this list consecutive vital points that are associated with the same hole vertex to satisfy the condition stated in the definition above. The resulting list induces a partition into $v$-pieces that is compatible with the convex partition. Since we did not restrict the convex partition in any way, this procedure enables us to find a compatible partition into $v$-pieces for all convex partitions.

Note that the partition into $v$-pieces generated as described in the preceding paragraph is valid, that is, there are no intersecting vital edges. To see this, consider any two consecutive (with respect to the perimeter) $p$-edges $\{(v_h, v), (v_h, v')\}$, $v_h \in V_h$, $v, v' \in V_p$. If no $p$-edge is incident to the perimeter between $v$ and $v'$, then $(v_h, v)$ and $(v_h, v')$ must belong to the same hole-free convex polygon in the partition. Hence the $v$-piece of $v_h$, which contains $v$ can extend to any vertex between $v$ and $v'$, including $v'$.

It is shown in [7] (the full version) that given $k$ hole vertices in a convex polygon $P$, there are at most $E = \max(2k-2, 1)$ $v$-pieces in the minimum convex partition. Note however that the precise number of $v$-pieces is not crucial for the complexity result, it only helps to refine it a little bit. A rougher approximation of its number would suffice to get similar asymptotic upper bounds.

In order to generate all possible partitions into $v$-pieces (which we have to search), we proceed as described in the following sections.

## 3.2 Preprocessing Phase

We consider all possible combinations of non-crossing edges going between hole vertices, i.e., all non-crossing super-graphs on the $k$ hole vertices. The total number of such graphs is at most $O((2^3 \cdot 59)^k \cdot k^{-6})$ [4, 18].

As shown pointed out above, it can be shown that given a convex pointgon $G$, at most 3 $v$-pieces incident to a hole vertex are sufficient achieve a convex partition (See [7] as well for further explanation). Therefore for each non-crossing graph on the $k$ hole vertices, we allocate a label $i \in \{1, 2, 3\}$ to each non-convex hole vertex, which is meant to indicate how many $v$-pieces that hole vertex would have in a convex partition if it can be constructed. There are no more than $3^k$ such labelings for any non-crossing graph.

Since we know the upper bound $E = \max(2k-2, 1)$ on the total number of $v$-pieces the $k$ hole vertices may have in an MWCP [7], we discard all labelings where the total number of $v$-pieces is greater that $E$. To process one labeling, we allocate unique names to each hole's $v$-pieces as follows: A $v$-piece name is a tuple $(v_h, x)$, where $v_h \in V_h$ is a hole vertex and $x \in \{a, b, c\}$ distinguishes between the (at most) three $v$-pieces the hole vertex $v_h$ has.

We then consider the arrangement of lines, such that for each pair of hole vertices we have a line containing them. We look at all intersections of this arrangement with the perimeter. There are $O(k^2)$ such intersections. Therefore the intersections of these lines with the perimeter partition the perimeter into $O(k^2)$ pieces. We will refer to each such piece as a 'topologically homogeneous perimeter piece' (or 'homogeneous piece' for short). Assume we let the homogeneous pieces of the perimeter $CH(P)$ to be $\{CH(P) = C_0, C_1, \ldots, C_\mu\}$ in counterclockwise order.

For each vital edge (see Definition 3) of a $v$-piece there can be $O(k^2)$ possibilities as to which homogeneous piece the vital edge should be incident to. However, since the $v$-pieces are contiguous, that is, neighbouring $v$-pieces share a vital point, it suffices to allocate one tag $C_j$, $j = 1, 2, \ldots, \mu$, to each $v$-piece name $(v_h, x)$, instead of one tag for each vital edge. We define that the tag $C_j$ represents the homogeneous piece the *most clockwise* vital edge of $(v_h, x)$ is incident to. To find the homogeneous perimeter piece the most *counterclockwise* vital edge of a $v$-piece is incident to, we retrieve the tag of the neighboring $v$-piece that shares the vital point this vital edge goes to.

A $v$-piece name assigned to a homogeneous piece $C_j$ is a tuple $((v_h, x), C_j)$. There are $O(k^{4k-4} \cdot 2^{2k-2})$ such assignments to be considered for a given labeling $L$, since there are at most $2k-2$ $v$-piece names and each $v$-piece name can be assigned to $O(k^2)$ homogeneous pieces.

For each homogeneous assignment to $v$-piece name, we then take an arbitrary point on the perimeter of each homogeneous piece and connect it to all hole vertices which should have a vital edge going to it. Let $E'$ be the set of edges (line segments) created in this way, for all homogeneous pieces. We do this to:
(1) Check for possible edge intersections. (We have no intersections if and only if the set $E'$ together with all edges of the non-crossing supergraph does not lead to any intersections.) If there are edge intersections, we discard the current labeling. (2) Find an ordering $\Phi$ of the $v$-piece names of the current labeling $L$, corresponding to the counterclockwise ordering in which the vital edges of the $v$-pieces will appear on the shrunk perimeter if a convex partition is to be constructed. This step takes $O(k \log k)$ time, because it is basically a sorting operation.

Before we start the dynamic programming, which determines whether it is possible to place the $p$-edges on the perimeter vertices we know: 1) How many $v$-pieces are associated to each hole vertex. 2) The homogeneous piece a vital edge of a $v$-piece should go to. 3) The counterclockwise ordering $\Phi$ of the $v$-piece names. 4) All edges connecting hole vertices.

## 3.3 Dynamic Programming Phase

For each ordering $\Phi$ of the $v$-piece names, we look at coherent subsequences of $1 \leq j \leq 2k-2$ $v$-piece names. We consider and solve each subsequence of $j$ $v$-piece names with the condition that either a hole vertex has all its $v$-piece names (at most three) in this subsequence, or none of them. We will refer to such subsequences as *valid coherent subsequences*. Let $C$ be a coherent piece of the polygon starting at perimeter vertex $l$ and containing $m$ vertices. If we have the most clockwise vital edge for a $v$-piece $i$ at vertex $l$ and the most counterclockwise vital edge for a $v$-piece $j$ at $(l+m) \bmod n$ (the other end of the chain), we want to find the optimal way to place the $p$-edges on the way counterclockwise from $l$ to $(l+m) \bmod n$ in order to minimise the length of the corresponding convex partition, if such a convex partition exists. To be precise, we specify each subproblem as a 4-tuple $(i, j, l, m)$, where $i \in 1 \ldots \max(2k-2, 1)$ indicates the position of the first element of the coherent subsequence in the counterclockwise ordered list of $v$-piece names, $j \in 1 \ldots \max(2k-2, 1)$ represents the number of $v$-piece names of the coherent subsequence, $l \in 0 \ldots n-k-1$ represents the vertex $v_l$ where the considered perimeter piece starts at, $m \in 1 \ldots n-k$ represents the number of vertices on the considered perimeter piece.

We start with smaller subproblems which are later used to solve larger subproblems, that is, we consider subproblems in the order of increasing $j$ and $m$.

For each subproblem we store the length of the minimum convex partition if it is possible to obtain a convex partition. Otherwise we store $\infty$ indicating a convex partition cannot be constructed.

See [7] for how the type of subproblems are solved.

## 3.4 Analysis

We considered all possible combinations of non-crossing super-graphs on the $k$ hole vertices. There are $O((2^3 \cdot 59)^k \cdot k^{-6})$ number of such non-crossing super-graphs [7]. It takes $O((2^3 \cdot 59)^k \cdot k^{-6})$ time to enumerate all such non-crossing super-graphs [7]. For each non-crossing super-graph $G'$, we considered all the possible labelings of the number of $v$-pieces corresponding to each hole vertex. There are not more than $3^k$ such labelings for each non-crossing super-graph. For each labeling $L$, we considered all possible homogeneous assignments. There are $O(k^{4k-4} \cdot 2^{2k-2})$ such assignments to be considered for a given labeling $L$. For each assignment, we then obtained the counterclockwise ordering $\Phi$ of the corresponding $v$-piece names in $O(k \log k)$ time. For the ordering $\Phi$ of an assignment $\Pi$ of a labeling $L$, we then check for possible edge intersections in time polynomial in $k$ and then run the dynamic programming algorithm which takes $O(n^3k^2)$ time at each time it is called.

This is because, the memory requirement in the worst case is dominated by the $O(n^2 \cdot k^2)$ space for the table entries. We solve directly each subproblem in time $O(n)$ with the help of the smaller subproblems. That is why each call of the dynamic programming algorithm takes $O(n^3k^2)$ time. We call the dynamic programming algorithm at most $O(k^{4k-10} \cdot 2^{13k})$ times. Thus the time taken to solve the MWCP problem is $O((2^3 \cdot 59)^k \cdot (k^{-6}) \cdot 3^k \cdot (k^{4k-4} \cdot 2^{2k-2}) \cdot n^3k^2) = O(n^3 \cdot k^{4k-8} \cdot 2^{13k})$.

It is straightforward to generalize this result to the case where we have as input a PSLG $G$ and $k$ is the total number of holes and/or reflex vertices inside the convex hull of $G$.[1]

## References

[1] P.K. Agrawal, E. Flato, and D. Halperin. Polygon Decomposition for Efficient Construction of Minkowski Sums. *Computational Geometry* 21(1-2):39–61. Elsevier Science, Amsterdam, Netherlands 2002

[2] O. Aichholzer, G. Rote, B. Speckmann, and I. Streinu. The Zigzag Path of a Pseudo-Triangulation. *Proc. WADS*, LNCS 2748:377–388. Springer-Verlag, Berlin, Germany 2003

[3] R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, NY, USA 1999

[4] A. Garcia, M. Noy, and J. Tejel. Lower Bounds on the Number of Crossing-free Subgraphs of $K_N$. *Computational Geometry, Theory and Applications* 16:211–221. Elsevier Science, Amsterdam, Netherlands 2000

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to Theory of NP-Completeness.* Freeman, New York, NY, USA 1979

[6] P.D. Gilbert. New Results in Planar Triangulations. *Report R-850.* University of Illinois 1979

[7] M. Grantson. *Fixed-Parameter Algorithms and Other Results for Optimal Convex Partitions.* Licentiate thesis, LU-CS-TR:2004-231, ISSN 1650-1276 Report 152. Lund University, Sweden 2004

[8] M. Grantson, C. Borgelt and C. Levcopoulos. A Fixed Parameter Algorithm for Minimum Weight Triangulation: Analysis and Experiments. Technical Report LU-CS-TR:2005-234, ISSN 1650-1276 Report 154. Lund University, Sweden 2005

[9] J. Keil. *Decomposing a Polygon into Simpler Components.* Ph.D. thesis (Report 163/8). Univ. of Toronto, Toronto, Canada 1983

[10] M. Hoffmann and Y. Okamoto. The Minimum Triangulation Problem with Few Inner Points. *Proc. IWPEC*, LNCS 3162:200–212. Springer-Verlag, Berlin, Germany 2004

[11] J. Keil. Decomposing a Polygon into Simpler Components. *SIAM Journal on Computing* 14:799–817. Society of Industrial and Applied Mathematics, Philadelphia, PA, USA 1985

[1]The complexity of our algorithm increases as if every reflex vertex were a hole vertex; i.e. $k$ would then be the sum of all holes and the number of reflex vertices on the outer perimeter.

[12] G.T. Klincsek. Minimal Triangulations of Polygonal Domains. *Annals of Discrete Mathematics* 9:121–123. ACM Press, New York, NY, USA 1980

[13] E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On Two-Dimensional Data Organization, Part I. *Fundaments Informaticae* 2:211–226. Polish Mathematical Society, Warsaw, Poland 1979

[14] A. Lubiw. The Boolean Basis Problem and How to Cover Some Polygons by Rectangles. *SIAM Journal on Discrete Mathematics* 3:98–115. Society of Industrial and Applied Mathematics, Philadelphia, PA, USA 1990

[15] D. Moitra. Finding a Minimum Cover for Binary Images: An Optimal Parallel Algorithm. *Algorithmica* 6:624–657. Springer-Verlag, Berlin, Germany 1991

[16] W. Mulzer and G. Rote. Minimum-weight Triangulation is NP-hard. *Proc. 22nd Ann. Symp. on Computational Geometry.* to appear, 2006

[17] D. Plaisted and J. Hong. A Heuristic Triangulation Algorithm. *Journal of Algorithms* 8:405–437. Academic Press, San Diego, CA, USA 1987

[18] F. Santos and R. Seidel. A Better Upper Bound on the Number of Triangulations of a Planar Point Set. *arXiv:math.CO/0204045 v2* 2002

# Pseudo-Convex Decomposition of Simple Polygons[*]

Stefan Gerdjikov[†] Alexander Wolff[‡]

## Abstract

We extend a dynamic-programming algorithm of Keil and Snoeyink for finding a minimum convex decomposition of a simple polygon to the case when both convex polygons and pseudo-triangles are allowed. Our algorithm determines a minimum pseudo-convex decomposition of a simple polygon in $O(n^3)$ time where $n$ is the number of the vertices of the polygon. In this way we obtain a well-structured decomposition with fewer polygons, especially if the original polygon has long chains of concave vertices.

## 1 Introduction

*Pseudo-triangles* are simple polygons with exactly three convex angles, i.e. interior angles of less than 180°. Recently they have emerged to have geometrical properties of interest for rigidity theory and ray-shooting problems [2]. This is why pseudo-triangles have been considered in relation with the *decomposition problem* of a set of points. It is defined as follows.

Given a set $S$ of $n$ points in the plane, decompose the convex hull of $S$ into polygons of a given type such that the vertices of the polygons are in $S$ and each point in $S$ is a vertex of at least one of the polygons. The decomposition is called *convex* if only convex polygons are allowed, *pseudo-triangulation* if only pseudo-triangles are allowed, and *pseudo-convex* if both pseudo-triangles and convex polygons can be used. Convex decompositions have been considered by Fevens et al. [3]. Streinu [7] shows that the minimum number of edges needed to obtain a pseudo-triangulation is $2n-3$ and thus, by Euler, the number of pseudo-triangles is $n-2$, which does not depend on the structure of the point set but only on its size. This motivates research on the problem of enumerating all minimum pseudo-triangulations [2]. Aichholzer et al. [1] study pseudo-convex decompositions. They show that each minimum pseudo-convex decomposition of a set of $n$ points consists of less than $7n/10$ polygons.

A related problem is the decomposition of *simple polygons* into convex polygons or pseudo-triangles, e.g. for point location or ray shooting. For decompositions of simple polygons the same terms as for decompositions of point sets apply. A decomposition is called *minimum* if it consists of the minimum number of regions.

In this paper we give an algorithm for computing minimum pseudo-convex decompositions of simple polygons. Given a simple polygon we use the same approach as Gudmundsson and Levcopoulos [4] to determine all geodesics in the polygon which can be sides of a pseudo-triangle and present a simple way to check whether three such geodesics form a pseudo-triangle. We use dynamic programming to solve proper subproblems which then can be combined to obtain a global solution. The resulting algorithm runs in $O(n^3)$ time and uses $O(n^2)$ space.

Our algorithm is based on a general technique for decomposing a simple polygon into polygons of a certain type proposed by Keil [5]. The technique is based on optimally decomposing subpolygons each of which is obtained from the original by drawing a single diagonal. This idea yields an $O(n^3 \log n)$-time algorithm for the *convex* decomposition problem [5]. Keil and Snoeyink [6] improve Keil's result by giving an $O(\min(nr^2, r^4))$-time algorithm, where $r$ is the number of reflex vertices of the polygon.

## 2 Characterization of Pseudo-Triangles

We use $P^+(A_i, A_j)$ and $P^-(A_i, A_j)$ to denote the paths on the boundary $\partial P$ from a vertex $A_i$ to a vertex $A_j$ of $P$ in clockwise and anticlockwise direction, respectively. With $\mathrm{vis}(A_i)$ we denote the list of all vertices of $P$ which are visible from $A_i$ in clockwise order starting with $A_{i+1}$. Unless stated otherwise, the vertices of a polygon will be given in clockwise order.

**Definition 1** *Let $P = A_0A_1 \dots A_{n-1}$ be a simple polygon. A path $p = B_1B_2 \dots B_m$ from $A_i$ to $A_j$ is a* concave geodesic *with respect to the polygon $P$ if it satisfies the following three conditions, see Fig. 1:*

*(G1) $B_1 = A_i$ and $B_m = A_j$.*

*(G2) For each $k < m$ it holds that $B_{k+1}$ is the last vertex on $P^+(B_k, A_j)$ which is visible from $B_k$.*

*(G3) $B_1B_2 \dots B_m$ is a convex, anticlockwise oriented polygon.*

[*]Work supported by grant WO 758/4-2 of the German Research Foundation (DFG).

[†]st_gerdjikov@abv.bg

[‡]Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany. WWW: i11www.ira.uka.de/people/awolff

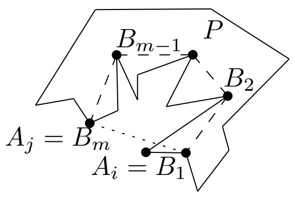


Figure 1: The geodesic $B_1B_2 \dots B_m$ from $A_i$ to $A_j$ is concave with respect to the simple polygon $P$.

**Remark 1** *If $B_1B_2 \dots B_m$ is a concave geodesic from $B_1$ to $B_m$ with respect to a simple polygon $P$ then $B_2 \dots B_m$ is a concave geodesic from $B_2$ to $B_m$ with respect to $P$.*

For our further considerations we will need the following fact [6]:

**Fact 1** *Let $A_i$ be a vertex of $P = A_0A_1 \dots A_{n-1}$. Then the cyclic order of the line segments $A_iA_j$ with $A_iA_j \subseteq P$ around $A_i$ is the same as the order of their other endpoints along $\partial P$.*

The following lemma states the relationship between the concave geodesics in a simple polygon and the pseudo-triangles that can participate in a decomposition of the polygon.

**Lemma 1** *If a pseudo-triangle $T$ is contained in a simple polygon $P = A_0A_1 \dots A_{n-1}$ with convex vertices at $A_j$, $A_k$ and $A_l$, $j < k < l$, then the paths $T^+(A_j, A_k)$, $T^+(A_k, A_l)$ and $T^+(A_l, A_j)$ are concave geodesics with respect to $P$.*

**Proof.** (Sketch) Due to symmetry it suffices to prove that $T^+(A_j, A_k)$ is a concave geodesic. Properties (G1) and (G3) of a concave geodesic obviously hold. Thus we have to verify only property (G2).

First note that $T^+(A_j, A_k)$ contains only vertices of $P$ that lie on $P^+(A_j, A_k)$, for otherwise $T$ wouldn't be simple. Now assume that $T^+(A_j, A_k)$ does not satisfy property (G2), see Fig. 2. Let $k > i \geq j$ be such that $A_i \in T^+(A_j, A_k)$ violates the construction proposed in property (G2). Let $A_s$ be the vertex on $T^+(A_j, A_k)$ after $A_i$ and let $k \geq t > s$ be such that $A_t$ is visible from $A_i$. It is clear that $s > i$. Due to Fact 1 we obtain that the edges $A_iA_{i+1}$, $A_iA_s$ and $A_iA_t$ appear in clockwise order around $A_i$. In particular, because of the convexity of $T^+(A_i, A_k)A_i$, $A_iA_t$ intersects $T^+(A_i, A_k)$ only in $A_i$ and $A_iA_s$ is contained in the polygon $P^+(A_i, A_t)A_i$. However, $A_k$ lies outside this polygon and thus $T^+(A_i, A_k)$ leaves $P^+(A_i, A_t)A_i$ in some point $x$ which does not belong to $A_iA_t$, see Fig 2. Therefore $T^+(A_i, A_k)$ leaves $P$. Contradiction. $\square$

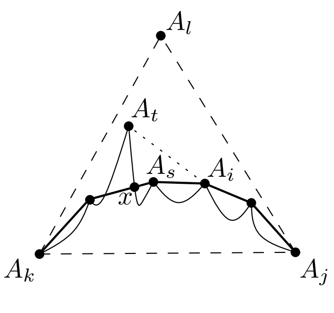


Figure 2: Each pseudo-triangle consists of three concave geodesics that connect its convex vertices. The arcs denote the boundary of $P^+(A_j, A_t)$ and the solid lines denote the edges of $T^+(A_j, A_k)$.

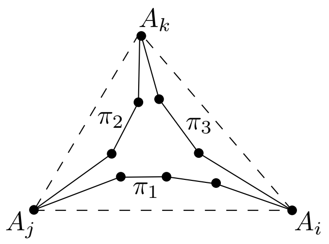


Figure 3: Testing whether three concave geodesics $\pi_1$, $\pi_2$, and $\pi_3$ define a pseudo-triangle

Next we establish the converse relation. Namely three concave geodesics determine a pseudo-triangle.

**Lemma 2** *Let $P = A_0A_1 \dots A_{n-1}$ be a simple polygon. Further let $i < j < k$ and $\pi_1 = A_i \dots A_j$, $\pi_2 = A_j \dots A_k$ and $\pi_3 = A_k \dots A_i$ be concave geodesics with respect to $P$. If the triangle $A_iA_jA_k$ is clockwise oriented, then the polygon $\pi_1\pi_2\pi_3$ is a pseudo-triangle.*

**Proof.** (Idea) See Fig. 3. Using that, say $\pi_1$ and $\pi_2$ have only one common vertex, one can show that they have no other common points. Then the orientation of the triangle $A_iA_jA_k$ together with property (G2) from Definition 1 provide that in fact $\pi_1$ is contained in the triangle $A_iA_jA_k$. Similar considerations for the paths $\pi_2$ and $\pi_3$ show that $\pi_1\pi_2\pi_3$ is a pseudo-triangle. $\square$

## 3 Algorithm

We use the same approach for finding a minimum pseudo-convex decomposition of a simple polygon as Keil and Snoeyink [6] for finding the minimum convex decomposition of a polygon. Namely we consider smaller simple polygons which are obtained from the original polygon by drawing a single diagonal. For each such polygon we make assumptions in what sort of polygon the diagonal can be included. In case the diagonal is a part of a convex polygon we use the algorithm of Keil and Snoeyink [6]. In case the diagonal is part of a pseudo-triangle we proceed as follows. Assume we have a precomputed list $L$ of all concave geodesics w.r.t. $P$. Then we can filter $L$ to find all pseudo-triangles that contain the diagonal as an edge. For each such pseudo-triangle $T$ we compute the size of an optimal decomposition that contains $T$. The optimal solution is the minimum of the solutions obtained in the two cases. Finally we apply dynamic programming, just as Keil and Snoeyink [6].

Now we describe our ideas in detail. Let $P = A_0A_1 \dots A_{n-1}$ be a simple polygon. We use definitions similar to those in [6]. If $i < j$ and $A_j$ is visible from $A_i$ in $P$ then we denote the line segment $A_iA_j$ by $d_{ij}$ and call it a *diagonal* of $P$. In particular each edge of $P$ is a diagonal. For each such diagonal a simple polygon $P_{ij} = A_iA_{i+1} \dots A_j$ is defined.

**Definition 2** *Let $\mathcal{D}$ denote the set of all pseudo-convex decompositions of a polygon $P_{ij}$. Then we introduce the following parameters:*

$$\begin{aligned} w_{ij} &= \min\{|D| : D \in \mathcal{D}\} \\ cw_{ij} &= \min\{|D| : D \in \mathcal{D}, \text{ the edge } d_{ij} \text{ is contained in a convex polygon}\} \\ pw_{ij} &= \min\{|D| : D \in \mathcal{D}, \text{ the edge } d_{ij} \text{ is contained in a pseudo-triangle }\} \end{aligned}$$

Clearly $w_{ij} = \min(cw_{ij}, pw_{ij})$.

Given the values $w_{kl}$ for each $k, l$ with $l - k < j - i$ and a list of all concave geodesics for the polygon $P$ we first describe how to find $pw_{ij}$. We consider all concave geodesics which contain the edge $A_iA_j$ and no vertex $A_k \in P$ with $k < i$ or $k > j$. For each such path $\pi_1 = B_1B_2 \dots B_m$ we go along $P^-(B_1, B_m)$ and for each vertex $A_l \in P^-(B_1, B_m)$ we check whether there exist concave geodesics $\pi_2 = B_m \dots A_l$ and $\pi_3 = A_l \dots B_1$. If $\pi_2$ and $\pi_3$ exist, we apply Lemma 2 to check whether the paths $\pi_1$,$\pi_2$ and $\pi_3$ determine a pseudo-triangle. If this is the case, an optimal decomposition of $P_{ij}$ contains this pseudo-triangle if and only if for each pair $(k, l) \neq (i, j)$ such that $A_kA_l$ is an edge of $\pi_1\pi_2\pi_3$ the polygon $P_{kl}$ is optimally decomposed.

Thus if $w(\pi)$ denotes the sum of all $w_{kl}$ where $A_kA_l$ lies on a geodesic $\pi$, then it is clear that the optimal decomposition of $P_{ij}$ using the pseudo-triangle $\pi_1\pi_2\pi_3$ consists of

$$s(\pi_1, A_l) = \sum_{A_kA_l \in \pi_1, A_kA_l \neq A_iA_j} w_{kl} + w(\pi_2) + w(\pi_3) + 1$$

polygons. Now we can compute $pw_{ij}$ as the minimum of $s(\pi_1, A_l)$ over all pairs $(\pi_1, A_l)$ that fulfill the above requirements.

To find the value $cw_{ij}$ we consider all vertices $A_k$ on the path $P^-(A_i, A_j)$ which are visible both from $A_i$ and $A_j$. If $A_iA_j$ is an edge of a convex polygon, then this polygon is either the triangle $T = A_iA_jA_k$ or a convex polygon $C = A_j \dots A_k \cup T$, where $A_j \dots A_k$ is a smaller convex polygon. In the former case, an optimal decomposition of $P_{ij}$ consists of $w_{ik} + w_{kj} + 1$ polygons. In the latter case the decomposition of $P_{ij}$ is the union of two pseudo-convex decompositions: (i) that of $P_{ik}$ and (ii) that of $P_{kj}$ under the condition that $A_kA_j$ is an edge of a convex polygon $C'$ with $C' \cup T$ convex. In (i) an optimal decomposition of $P_{ik}$ consists of $w_{ik}$ polygons. To determine an optimal decomposition of $P_{kj}$ in (ii) we use the approach of Keil and Snoeyink [6], which relies on the following observation.

We call a *diagonal-convex* decomposition of $P_{ij}$ a decomposition where $d_{ij}$ is the diagonal of a convex polygon. For each polygon $P_{ij}$ we store not only the value $cw_{ij}$ but also a list $CL_{ij}$ of *representatives* of diagonal-convex decompositions of $P_{ij}$ which attain $cw_{ij}$. Given an optimal diagonal-convex decomposition $\Delta$ of $P_{ij}$ the representative $(s, t)$ of $\Delta$ is uniquely defined by a pair of vertices $\{A_s, A_t\} \cap \{A_i, A_j\} = \emptyset$. More precisely, $A_s$ and $A_t$ are those vertices of $P_{ij}$ that are adjacent to $A_i$ and $A_j$, respectively, in the only polygon $\Pi \in \Delta$ with $d_{ij}$ being an edge of $\Pi$. We store only representatives $(s, t)$ satisfying the property that for each other representative $(s', t') \neq (s, t)$ of $P_{ij}$ either $s > s'$ or $t < t'$. Using the same arguments as Keil and Snoeyink [6, Section 3], one can show that in $O(n)$ time the value $cw_{ij}$ can be correctly determined and the list $CL_{ij}$ can be constructed—provided the lists $CL_{kj}$ are available for all $i < k < j$.

## 4 Complexity

We now investigate the complexity of our algorithm. We first modify slightly Theorem 2 in [4].

**Proposition 3** *Given a simple polygon $P = A_0A_1 \dots A_{n-1}$ we can construct in $O(n^2)$ time a data structure such that for any pair $(i, j)$ it can decide in $O(1)$ time whether there is a concave geodesic $\pi$ from $A_i$ to $A_j$. If $\pi$ exists, the data structure provides an $O(l)$-time walk along $\pi$, where $l$ is the length of $\pi$.*

**Proof.** We first compute all lists vis($A_i$) in $O(n^2)$ total time. Then we use dynamic programming to check whether there is a concave geodesic $\pi$ from $A_i$ to $A_j$. If $\pi$ exists, we also compute the second and the second last vertex on $\pi$. We can walk on $\pi$ by repeatedly jumping to the second vertex of the remaining path, which by Remark 1 is also a geodesic.

We consider the pairs $(i, j)$ in increasing order of the number of vertices on the path $P^+(A_i, A_j)$. The edges $A_iA_{i+1}$ obviously correspond to concave geodesics and it is easy to determine the second and second last vertex of these paths.

When the length of $P^+(A_i, A_j)$ is greater than 1 we use the list vis($A_i$) to find the last vertex visible from

$A_i$ on $P^+(A_i, A_j)$—this is either $A_j$ or the last vertex visible from $A_i$ on $P^+(A_i, A_{j-1})$. Fact 1 allows us to handle vis$(A_i)$ in $O(1)$ time to obtain the desired information. Once we have found the last vertex $A_l$ visible from $A_i$ on $P^+(A_i, A_j)$ we check whether there is a concave geodesic $\pi$ from $A_l$ to $A_j$. If this is the case we use the second and the second last vertex on $\pi$ to check whether $A_i$ can be added to $\pi$ without violating property (G2). According to Remark 1 this is the only way for obtaining a concave geodesic from $A_i$ to $A_j$. Finally the second and the second last vertex on this path can also be computed in $O(1)$ time. Thus we need only $O(1)$ time per pair $(i, j)$ in order to check whether there exists a concave geodesic from $A_i$ to $A_j$ and—in case it does—to find the second and the second last vertex on this path. Because the number of all pairs $(i, j)$ is $O(n^2)$, this results in an $O(n^2)$-time algorithm with the desired properties. □

Notice that in the proof of Proposition 3 we can compute also the vertices with the greatest and smallest indices that lie on a given concave geodesic $\pi$ without increasing the complexity of the algorithm. Moreover, we can check in constant time whether these two vertices are adjacent on $\pi$. We use this observation to compute a list $PL_{ij}$ for each diagonal $d_{ij}$. In this list we store all pairs $(k, l)$ such that there is a concave geodesic from $A_k$ to $A_l$ which contains $d_{ij}$ but no vertex with index smaller than $i$ or greater than $j$.

**Theorem 4** *The number of polygons in a minimum pseudo-convex decomposition of a simple polygon $P = A_0A_1 \dots A_{n-1}$ can be computed in $O(n^3)$ time.*

**Proof.** We first set up the data structure of Proposition 3 and compute the lists $PL_{ij}$. This takes $O(n^2)$ total time. Then we implement the algorithm of Section 3. Using the technique of Keil and Snoeyink [6] the computation of all $cw_{ij}$ can be carried out in total $O(n^3)$ time. To bound the time needed for the computation of $pw_{ij}$ first note that each concave geodesic $\pi = B_1 \dots A_jA_i \dots B_m$ is contained in at most one list $PL_{ij}$ and thus it is considered once only. We walk along $\pi$ to determine the sum of the values $w_{kl}$ over all $(k, l) \neq (i, j)$ with $A_kA_l \subseteq \pi$. This takes $O(n)$ time according to Proposition 3. Then for each point $A_l$ on $P^+(B_m, B_1)$ we check whether there is a concave geodesic $\pi_1$ from $A_l$ to $B_1$ and a concave geodesic $\pi_2$ from $B_m$ to $A_l$. If this is the case, we use Lemma 2 to check whether $\pi\pi_1\pi_2$ is a pseudo-triangle. This takes $O(1)$ time. Finally we need the values $w(\pi_1)$ and $w(\pi_2)$ which can be computed in $O(n)$ time the first time we need them. Thus we walk along each geodesic only once and perform only $O(n)$ operations for each geodesic. The total number of concave geodesics is $O(n^2)$ which results in $O(n^3)$ time for determining all values $pw_{ij}$. Thus the number of polygons in a minimum pseudo-convex decomposition of a simple polygon $P$ with $n$ vertices can be computed in $O(n^3)$ time. □

## Acknowledgments

We thank the anonymous referee whose comments helped us to improve the readability of this paper.

## References

[1] O. Aichholzer, C. Huemer, S. Renkl, B. Speckmann, and C. D. Tóth. On pseudo-convex decompositions, partitions, and coverings. In *Proc. 21st European Workshop on Computational Geometry (EWCG'05)*, pages 89–92, Eindhoven, 2005.

[2] O. Aichholzer, G. Rote, B. Speckmann, and I. Streinu. The zigzag path of a pseudo-triangulation. In *Proc. Workshop on Algorithms and Data Structures (WADS'03)*, pages 377–388, 2003.

[3] T. Fevens, H. Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1–2):95–107, 2001.

[4] J. Gudmundsson and C. Levcopoulos. Minimum weight pseudo-triangulations. In K. Lodaya and M. Mahajan, editors, *Proc. 24th Int. Conf. Foundations Software Tech. Theoretical Comput. Sci. (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 299–310. Springer-Verlag, 2004.

[5] J. M. Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14:799–817, 1985.

[6] J. M. Keil and J. Snoeyink. On the time bound for convex decomposition of simple polygons. *Int. J. Comput. Geometry Appl.*, 12(3):181–192, 2002.

[7] I. Streinu. A combinatorial approach to planar non-colliding robot arm planning. In *Proc. 41st Annu. IEEE Sympos. Found. Comp. Sci. (FOCS'00)*, pages 443–453, 2000.

# The Existence of a Pseudo-triangulation in a given Geometric Graph

André Schulz*

## Abstract

We show that the problem of deciding if a pseudo-triangulation is contained inside a geometric graph is NP-complete. For this we investigate the Triangulation Existence Problem, which is known to be NP-complete. We present a new proof for its NP-completeness and modify it in such a way that it can be applied for pseudo-triangulations.

## 1 Introduction

A *pseudo-triangle* is a polygon with exactly three convex corners. A planar partition of a point set into pseudo-triangles is called *pseudo-triangulation.* A pseudo-triangulation is called pointed, if all its vertices are incident to an angle greater than $\pi$. Many different applications for pseudo-triangulations are known.

The main focus of this paper lies on the problem, if there exists a pseudo-triangulation as a subset of a given geometric graph. The geometric graph does not have to be planar. We call this problem the Pseudo-Triangulation Existence Problem (PTRI).

The "triangulation version" of the PTRI is known as the Triangulation Existence Problem (TRI). Lloyd showed in 1977 [6] that the TRI is NP-hard by a reduction from CNF-SAT. Although the idea behind the construction is not difficult, it seems hard to modify the complex gadgets for new NP-completeness results.

In section 2 we will present a new proof for TRI. Instead of reducing from CNF-SAT we will use a reduction from Planar 3-SAT(which was not known in 1977). This allows us a simpler construction for the NP-hardness proof. We profit from the fact that Planar 3-SATis more structured than CNF-SAT. As a side effect it is now easier to modify the construction in such a way that we can apply it for PTRI.

Pseudo-triangulations do not always show the same behavior as triangulations (to name just one example, their flipping distance is smaller [1]). It is a natural question, in how far known results from triangulations can be generalized for pseudo-triangulations. This might help to understand the properties of pseudo-triangulations better.

In [8] a NP-completeness result for triangulation (minimum vertex degree) was generalized for pseudo-triangulations with similar ideas used in this paper.

The problem of finding a pseudo-triangulation inside a triangulation was discussed in [7]. The complexity status of finding a pseudo-triangulation with minimal number of edges inside a triangulation was introduced as an open problem. Its status is still open.

## 2 A new proof for the NP-hardness of TRI

First of all we state the problem, for which we want to prove its NP-completeness

**Triangulation Existence Problem (TRI)**
**Input:** A geometric graph $G = (V, E)$.
**Question:** Is there a graph $G' = (V, E' \subset E)$ and $G'$ is a triangulation ?

**Theorem 1** *TRI is strongly NP-complete.*

The proof of the theorem will be given by the following discussion. The Problem lies in NP, because we can verify in polynomial time if a guessed subgraph of $G$ is a triangulation. To show NP-hardness, we reduce Planar 3-Sat to TRI. A formula $\phi$ is planar if it can be represented as a planar graph $G(\phi) = (V_\phi, E_\phi)$. The set $V_\phi$ is given by the variables and the clauses of $\phi$. The pairs of all (negated) variables and their associated clauses define $E_\phi$. The Problem if a planar formula in 3-CNF is satisfiable is known to be NP-complete [5].

The reduction from Planar 3-Sat is done by substituting edges and vertices of $G_\phi$ by more complex subgraphs (called *gadgets*). The resulting graph contains a triangulation, if and only if the formula is satisfiable,

We are using 4 different types of gadgets. The most essential gadget is the Wire gadget. It is responsible for carrying the value of a variable to the clauses and therefore it will be a replacement for the edges. The variables themselves are represented as a piece of the Wire gadget. To evaluate the clauses we introduce a Nand gadget and a Not gadget, which will be also used to negate variables. Finally we present a gadget which will split an edge, while maintaining the status of the wire for the outgoing parts. This gadget is called the Split gadget. Starting with the Wire gadget we will explain the gadget one by one.

*Institut für Informatik, Freie Universität Berlin, Germany, schulza@inf.fu-berlin.de

**Wire**

The Wire gadget represents the state of a literal which can be either true or false. Therefore it consist of a graph which contains exactly two triangulations (Figure 1). The two contained triangulations are called *black* (representing the value TRUE) and *dashed* (representing the value FALSE). We will also call their edges black and dashed. In the figures the dashed triangulation is drawn with dashed lines. The reader should check that it is not possible to switch inside the gadget from the dashed to the black triangulation and vice versa. This is due to the fact that there is no triangle which contains a dashed edge and a black edge. Therefore the choice of one edge determinates the status of all the other edges inside the gadget. It is possible to bend the gadget without destroying its structure. Figure 2 shows a 90 degree bend. A variable will be realized as a part of the Wire gadget (the part where we chose the fist diagonal).

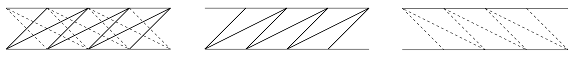


Figure 1: (a) The Wire (a); (b) & (c) its triangulations.

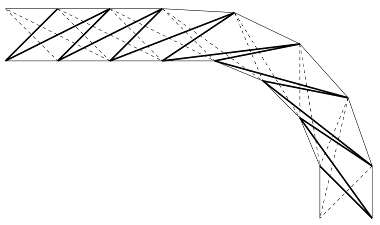


Figure 2: A Wire with a 90 degree bend.

**Split**

The Split gadget has three input parts (Figure 3). Like the Wire it contains a dashed and a black triangulation. Since there is no triangle with a dashed and a black edge, it is not possible to switch between the black and dashed triangulation inside the gadget. For this reason the status of the wire is the same on the three output parts. The gadget will be used for producing multiple copies of a variable (or its negated version). Its open ends connect perfectly to the Wire.

**Not**

To realize a negation, we have to change the orientation of the diagonals inside the wire. Figure 4 shows how this can be done. Again we have two triangulations induced by the gadget and there exists no triangle with dashed and black edges. Hence, the orientation of the diagonals is switched by the gadget.

**Nand**

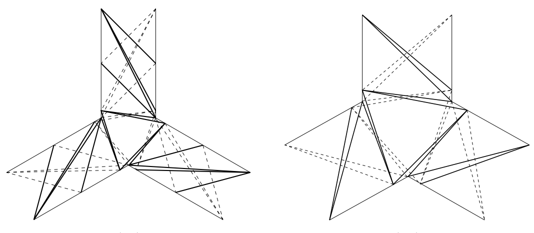


Figure 3: (a) The Split gadget; (b) a close up of the gadget.

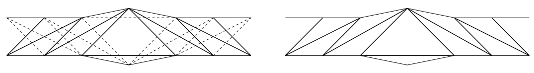


Figure 4: (a) The Not gadget; (b) one of its triangulation.

The last gadget needed for the reduction is the Nand gadget. Its purpose is to evaluate the clauses (we can simulate an OR gate).

The Nand has three inputs and is slightly more complex than the other gadgets. It allows more than two triangulations. Therefore it contains edges which can not be handled as black or dashed (we call these edges gray). Lets assume that the dashed triangulation represents the value *true*. The gadget allows a triangulation for all possible input combinations, except when all are dashed triangulations. Lets have a closer look at the gadget shown in Figure 5. We see that three input wires meet in a 9-gon, which is filled with diagonals. The structure is symmetric under rotation by 120 degrees, but it is not symmetric by reflection. It can be observed that there are only three gray diagonals crossing the black edge $a_1a_2$ but there are 6 gray edges crossing the dashed diagonal $b_1b_2$. Hence having three dashed triangulated input wires makes it impossible to find a triangulation of the 9-gon. The removal of all gray diagonals in this setting leaves an empty hexagon (Figure 6.a), which

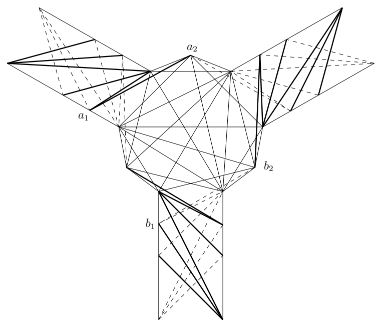


Figure 5: The Nand gadget.

can't be triangulated. On the other hand, all other combination of the input can be triangulated (as seen in Figure 6.b-d). All other combinations are symmetric versions of Figure 6. Therefore the functionality of a NAND gate is provided by the gadget and clauses can be evaluated in combination with the NOT gadget.

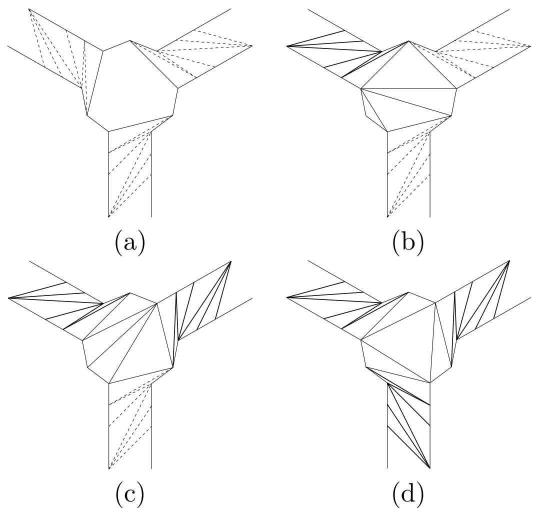


Figure 6: The NAND gadget with different input values.

After the substitution of the edges and vertices of $G_\phi$ by the gadgets, we might have pockets and holes inside the resulting graph. We will triangulate them arbitrarily. Any edge of the graph which is not crossed by any other edge has to be contained in the triangulation. Thus no boundary edge of a gadget can be deleted and the triangulation of the holes doesn't affect the functionality of the gadgets.

Clearly a formula $\phi$ is satisfiable, if and only if there exists a triangulation inside the constructed graph. To finishes the proof of Theorem 1 we observe that the reduction can be made in polynomial time.

## 3 Pseudo-triangulations inside a graph

The new proof of the NP-completeness of TRI allows us to attack similar problems with the same basic idea. One natural variation of TRI is the following:

**Pseudo-Triangulation Existence Problem (PTRI)**

**Input:** A geometric graph $G = (V, E)$.

**Question:** Is there a graph $G' = (V, E' \subset E)$ and $G'$ is a pointed pseudo-triangulation ?

**Theorem 2** *PTRI is strongly NP-complete.*

As done in the proof for Theorem 1 the proof will be given in the following discussion. Like in Section 2, we reducing again from PLANAR 3-SAT and we will introduce again the same set of gadgets (namely WIRE, SPLIT, NOT and NAND).

**Wire**

The WIRE gadget can be easily obtained from the one used for TRI. Now two pseudo-triangulations are part of the gadget. We call them again *dashed* and *black*. It is not possible to find a pseudo-triangle inside the gadget, which consists of a dashed and a black diagonal. Hence the choice of a diagonal determinates the whole pseudo-triangulation. See Figure 7 for the corresponding pictures. It should be clear that bending

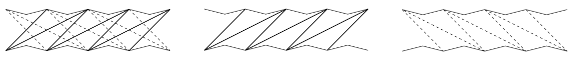


Figure 7: The WIRE and its pseudo-triangulations.

the gadget is no problem. We omit the picture for the 90 degree bend.

**Not**

The NOT gadget is basicly the same as the one for the triangulation case. We just ensure that every vertex contains an angle greater than $\pi$. Figure 8 shows the gadget.

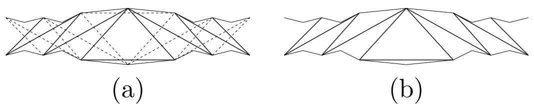


Figure 8: (a) The NOT gadget; (b) one of its triangulation.

**Split**

The SPLIT is shown in Figure 9.a. It has three incoming wire parts and a central area, which has to be covered by a pseudo-triangle. This is possible, if all

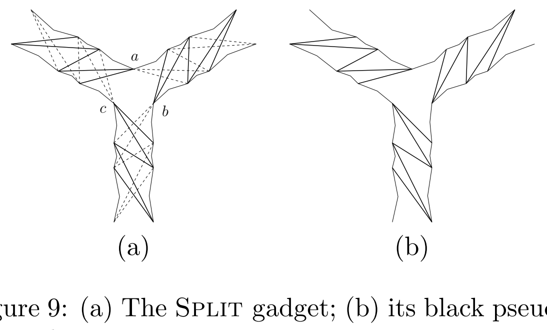


Figure 9: (a) The SPLIT gadget; (b) its black pseudo-triangulation.

three wires are dashed or black pseudo-triangulations (Figure 9.b shows the black pseudo-triangulation of the gadget). It is not possible to find a pseudo-triangulation for any other combination. In these case, we could not construct an angle greater $\pi$ at at least one of the points $a, b, c$. Therefore the face covering the center of the gadget would be at least a pseudo-quadrilateral.

**Nand**

The NAND gadget (Figure 10) is based on the gadget used for TRI. Again we have a 9-gon and a number of gray diagonals. These have to be used to pseudo-triangulate the 9-gon. The pseudo-triangulation of the incoming wires forbids a certain set of diagonals, depending if the pseudo-triangulation is black or dashed. If all wires use dashed pseudo-triangulations it is not possible to pseudo-triangulate the 9-gon (Figure 11.a). All other cases (shown in Figure 11.b-d, up to symmetric equivalences) can produce valid pseudo-triangulations.

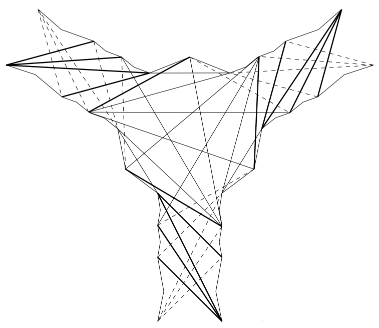


Figure 10: The NAND gadget.

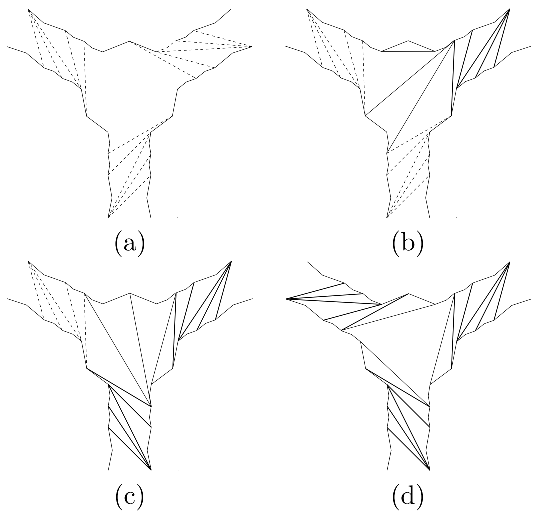


Figure 11: The NAND gadget with different input values.

The remaining holes of the constructed graph will be arbitrarily pseudo-triangulated (which is always possible). It remains to show that gadget boundary edges must be part of the pseudo-triangulation if one exists. If all vertices are pointed this follows from the fact that the removal of an edge, which is not crossed by any other edge, will construct a pseudo-quadrilateral. We leave the discussion for the non-pointed vertices of the gadgets to the full version of the paper.

It follows that this set of gadgets presents a valid reduction from PLANAR 3-SAT to PTRI.

## 4 Remarks and Open Problems

Since the gadgets for the reduction to TRI are small and easy to understand, they can be used to prove several similar NP-completeness results. One might think of the Problem if a quadrilateralization is contained inside given geometric graph.

Another interesting question related to PTRI is the following.

**Planar Rigid Graph Containment (PRGC)**
**Input:** Geometric graph $G = (V, E)$.
**Question:** Is there a planar (minimal) rigid graph $G' = (V, E' \subset E)$?

Although there are fast algorithms for testing planarity [4] and rigidity (e.g. [2]) it is not clear if we can find efficiently a rigid planar subset of a given graph. This problem is related to PTRI, since every pointed pseudo-triangulation forms a planar minimal rigid graph. Furthermore every planar minimal rigid graph can be embedded as a pointed pseudo-triangulation [3]. The gadgets we used will not help us, since a different embedding will destroy their functionality.

## References

[1] S. Bereg. Transforming pseudo-triangulations. *Information Processing Letters*, 90(3):141–145, 2004.

[2] A. R. Berg and T. Jordán. Algorithms for graph rigidity and scene analysis. In *ESA*, volume 2832 of *Lecture Notes in Computer Science*, pages 78–89. Springer, 2003.

[3] R. Haas, G. Rote, D. Orden, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, and W. Whiteley. Planar minimally rigid graphs and pseudo-triangulations. *Computational Geometry, Theory and Applications*, 2005.

[4] J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[5] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[6] E. L. Lloyd. On triangulations of a set of points in the plane. In *Proc. 18th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 228–240, 1977.

[7] G. Rote, C. A. Wang, L. Wang, and Y.-F. Xu. On constrained minimum pseudotriangulations. In *COCOON*, volume 2697 of *Lecture Notes in Computer Science*, pages 445–454. Springer, 2003.

[8] A. Schulz. New results on pseudo-triangulations with low vertex degree. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG'05)*, pages 130–133, 2005.

# Ray Shooting Amidst Fat Convex Polyhedra in 3-Space

Boris Aronov[*] Mark de Berg[†] Chris Gray[†]

**Abstract**

We present a data structure for ray-shooting queries in a set of disjoint convex fat polyhedra of total complexity $n$ in $\mathbb{R}^3$. The data structure uses $O(n^{2+\varepsilon})$ storage and preprocessing time, and queries can be answered in $O(\log^2 n)$ time. A trade-off between storage and query time is also possible: for any $m$ with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon})$ storage and preprocessing time such that queries take $O((n/\sqrt{m}) \log^2 n)$ time.

## 1 Introduction

The ray-shooting problem is to preprocess a set $\mathcal{P}$ of objects in $\mathbb{R}^d$ for the following queries: what is the first object (if any) in $\mathcal{P}$ that is hit by a query ray? Such queries form the basis of ray-tracing algorithms, and they can be used to approximate form factors in radiosity methods. Since ray shooting is an integral part of many graphics applications, it should not be surprising that it has received much attention, both in computer graphics and computational geometry. In fact, after the range-searching problem it is probably one of the most widely studied data-structuring problems in computational geometry. The survey by Pellegrini [12] and the book by De Berg [3] discuss several of the data structures that have been developed within computational geometry for the ray-shooting problem (although there is also much work that is not covered there, for example, research concerning ray shooting in 2-dimensional scenes, or in $d$-dimensional space for $d > 3$). In the discussion below, we will restrict our attention to results on ray shooting in $\mathbb{R}^3$. Furthermore, we focus on the general ray-shooting problem, where the origin and direction of the query ray are unrestricted.

If the set $\mathcal{P}$ consists of $n$ arbitrary triangles, the best known structures with $O(\log n)$ query time use $O(n^{4+\epsilon})$ storage [3, 11], whereas the best structures with near-linear storage have roughly $O(n^{3/4})$ query time [2]. More generally, with $O(m^{1+\varepsilon})$ storage, for any $m$ with $n < m < n^4$, one can obtain $O((n/m^{1/4}) \log n)$ query time using $O(m^{1+\varepsilon})$ storage [2]. Better results have been obtained for several special cases. When the set $\mathcal{P}$ is a collection of $n$ axis-parallel boxes, one can achieve $O(\log n)$ query time with a structure using $O(n^{2+\varepsilon})$ storage [3]. Again, a trade-off between query time and storage is possible: with $O(m^{1+\varepsilon})$ storage, for any $m$ with $n < m < n^2$, one can achieve $O((n/\sqrt{m}) \log n)$ query time using $O(m^{1+\varepsilon})$ storage. If $\mathcal{P}$ is a set of $n$ balls, then it is possible to obtain $O(n^{2/3})$ query time with $O(n^{1+\varepsilon})$ storage [14], or $O(n^\varepsilon)$ query time with $O(n^{3+\varepsilon})$ storage [10].

Both axis-parallel boxes and balls are very special objects, and in most graphics applications the scene will not consist of such objects. The question thus becomes: is it possible to improve upon the ray-shooting bounds for arbitrary triangles for classes of objects that are more general than axis-parallel boxes or spheres? This is the problem we tackle in this paper. More precisely, we study the ray-shooting problem for disjoint convex polyhedra that are *fat*—see Section 2 for a formal definition.

**Related work.** Given the prominence of the ray-shooting problem and the interest in efficient algorithms and data structures for fat objects and other realistic input models in the past decade, it is not surprising that the ray-shooting problem for fat objects has been studied already. The results achieved so far are, however, quite limited. Most of the work on ray shooting among fat objects has dealt with shooting rays in a fixed direction [4, 5, 8]. When it comes to arbitrary rays, there are only a few results. For the case of *horizontal* fat triangles, there is a structure that uses $O(n^{2+\varepsilon})$ storage and has $O(\log n)$ query time [3], but the restriction to horizontal triangles is quite severe. Another related result is by Mitchell *et al.* [9]. In their solution, the amount of storage depends on the so-called *simple-cover complexity* of the scene, and the query time depends on the simple-cover complexity of the query ray. Unfortunately the simple-cover complexity of the ray—and, hence, the worst-case query time—can be $\Theta(n)$ for fat objects. In fact, this can happen even when the input is a set of cubes. The first (and so far only, as far as we know) result that works

[*]Department of Computer and Information Science, Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201-3840 USA; http://cis.poly.edu/~aronov. Research supported in part by NSF grant ITR-0081964 and by a grant from the US-Israel Binational Science Foundation. Part of the research was performed when B.A. visited TU Eindhoven in June 2005.

[†]Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands. Email: {mdberg,cgray}@win.tue.nl. This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

for arbitrary rays and rather arbitrary fat objects was recently obtained by Sharir and Shaul [13]. They present a data structure for ray shooting in a collection of fat triangles that has $O(n^{2/3+\varepsilon})$ query time and uses $O(n^{1+\varepsilon})$ storage. Curiously, their method does not improve the known bounds at the other end of the query-time–storage spectrum, so for logarithmic-time queries the best known storage bound is still $O(n^{4+\varepsilon})$.

**Our results.** We present a data structure for ray shooting with arbitrary rays in a collection $\mathcal{P}$ of disjoint convex fat polyhedra with $n$ vertices in total. Our structure requires $O(n^{2+\varepsilon})$ storage and has query time $O(\log^2 n)$. A trade-off between storage and query time is also possible: for any $m$ with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon})$ storage and has $O((n/\sqrt{m})\log^2 n)$ query time. Thus we improve upon the result by Sharir and Shaul in two ways: we reduce the query time for near-linear storage from $O(n^{2/3+\varepsilon})$ to $O(\sqrt{n}\log^2 n)$ and improve the bounds at the other end of the spectrum as well.

Of course, the two settings are not the same: Sharir and Shaul consider fat triangles, whereas we consider fat polyhedra. Indeed, our solution makes crucial use of the fact that fat polyhedra have a relatively large volume. Note that neither setting implies the other: fat triangles need not form fat polyhedra, and fat polyhedra do not necessarily have fat facets.

Our solution is based on the following idea. For each polyhedron $P$ we construct a constant number of so-called "towers" that lie inside $P$ and together cover the boundary of $P$. The towers are in some canonical form, which makes it easy to detect intersections of such a tower with a line segment. We believe that this technique, described in detail in Section 3, is of independent interest, and we expect it will find other applications in problems on fat polyhedra.

## 2 Preliminaries

**Definition and basic properties of fat objects.** We will use the definition of fatness introduced by Van der Stappen [15]. For a 3-dimensional object $o$, we use $\mathrm{vol}(o)$ to denote its volume.

**Definition 1** *An object $o$ is* $\beta$-fat *if for any ball $b$ whose center lies in $o$ and which does not completely contain $o$,* $\mathrm{vol}(b \cap o) \geq \beta \cdot \mathrm{vol}(b)$.

It is well known that any fat convex object $o$ admits two concentric balls, one containing $o$ and one contained in $o$, whose size ratio is bounded. Here we need a similar property, but for cubes instead of balls. For a cube $C$, let $\mathrm{size}(C)$ be the edge-length of $C$.

**Lemma 1** *Let $\sigma := \lceil 54\sqrt{3}/\beta \rceil$. For any convex $\beta$-fat object $o$ in $\mathbb{R}^3$, there exist concentric axis-aligned cubes $C^-(o)$ and $C^+(o)$ with $C^-(o) \subseteq o \subseteq C^+(o)$ such that $\frac{\mathrm{size}(C^+(o))}{\mathrm{size}(C^-(o))} = \sigma$.*

**Ray shooting and parametric search.** Agarwal and Matoušek [1] described a technique that reduces the ray-shooting problem on a set $\mathcal{P}$ of objects to the segment-emptiness problem, *i.e.*, testing whether a query segment intersects any of the objects in $\mathcal{P}$. Since then their technique has been used in several papers dealing with ray shooting [10, 13, 14]. We will also use this technique. In our setting, it implies the following: if we have a data structure for segment-emptiness queries, then we can use that same structure for ray-shooting queries at the cost of an extra (multiplicative) $O(\log n)$ factor in query time.

## 3 The data structure

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be the set of convex fat polyhedra that we wish to preprocess for ray-shooting queries. We use $n$ to denote the total number of vertices of the polyhedra. Our global strategy is roughly as follows.

We first present a decomposition of the boundary of each polyhedron into a constant number of pieces that are monotone in some canonical direction. Each such piece is extended into the polyhedron to obtain an object which we will call a *tower*. Next, we present a data structure to efficiently perform segment-emptiness queries on the towers. Using Agarwal and Matoušek's parametric-search technique mentioned above, we then convert this structure into a structure for ray shooting.

**The decomposition.** We first define the canonical directions that we will use in our decomposition. Let $C^+$ and $C^-$ be two concentric axis-aligned cubes such that $\frac{\mathrm{size}(C^+)}{\mathrm{size}(C^-)} = \sigma$, where $\sigma$ is defined as in Lemma 1. Since $\sigma$ is an integer, we can partition each face of $C^+$ into $\sigma^2$ squares of the same size as the facets of $C^-$. We use this to define a set $\mathcal{D}$ of $O(1/\beta^2)$ canonical directions, as follows. For each square $s$ on the top facet of $C^+$, we add to $\mathcal{D}$ the direction into which the top facet of $C^-$ must be translated to make it coincide with $s$. The remaining five facets of $C^+$ are treated similarly. The resulting set $\mathcal{D}$ of canonical directions has size $6\sigma^2 = O(1/\beta^2)$.

Next, we define the towers. A *tower in the direction* $\vec{d} \in \mathcal{D}$ is a convex polyhedron $t$ with the following properties:

(i) One of the facets of $t$ is an axis-parallel square; this facet if called the *base* of $t$ and it is denoted by $\mathrm{base}(t)$. We require that the orientation of

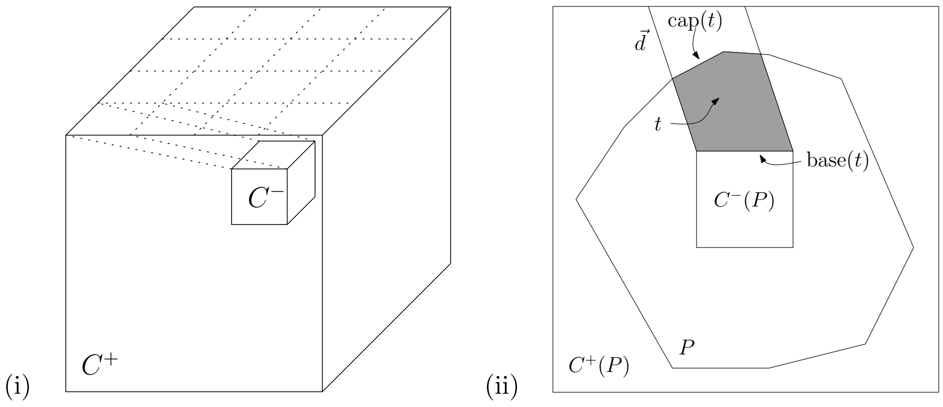


Figure 1: (i) Swept volume defining a tower. (ii) Two-dimensional analogue of a tower $t$.

the base—whether it is parallel to the $xy$-plane, to the $xz$-plane, or to the $yz$-plane—be uniquely determined by the direction $\vec{d}$. Hence, all towers in a given direction $\vec{d}$ have parallel bases.

(ii) The remaining facets of $t$ form a terrain in direction $\vec{d}$, that is, any line parallel to $\vec{d}$ and intersecting the base intersects the remaining facets either in a single point or in a line segment. We call the union of these remaining facets, excluding facets parallel to $\vec{d}$, the *cap* of the tower, denoted cap$(t)$.

Let $P \in \mathcal{P}$ be a $\beta$-fat convex polyhedron. The decomposition of $P$ is performed in a manner similar to the way we constructed the canonical directions. Let $C^-(P)$ and $C^+(P)$ be cubes with the properties given in Lemma 1. Partition each facet of $C^+(P)$ into $\sigma^2$ equal-sized squares. For each such square $s$ we construct a tower by sweeping $s$ towards the corresponding facet of $C^-(P)$, and taking the intersection of the swept volume and the polyhedron $P$. This way we obtain for each polyhedron $P$ one tower for each of the $|\mathcal{D}|$ canonical directions. We denote the set of towers constructed for $P$ by $T(P)$. The union of the towers in $T(P)$ is contained in $P$; the boundary of this union consists of the boundaries of $P$ and of $C^-(P)$.

**Testing for segment emptiness.** Before we describe the data structure for segment-emptiness queries, we describe necessary and sufficient conditions for a segment to intersect a polyhedron $P$. In the lemma below and in the rest of the paper, whenever we speak of "above" and "below" when referring to a specific tower, this is always with respect to the canonical direction $\vec{d}$ of that tower. More precisely, we say that a object $o$ is *below* an object $o'$ whenever there exists a (directed) line with orientation $\vec{d}$ that first intersects $o$ and then $o'$. A point is inside a tower, for instance, if and only if it is above the base and below the cap. Finally, we use proj$(o)$ to denote the projection of an object $o$ in direction $\vec{d}$ onto a plane orthogonal to $\vec{d}$.

**Lemma 2** *A segment $s = \overline{pq}$ intersects a polyhedron $P \in \mathcal{P}$ if and only if one of the following conditions holds:*

1. *$p$ or $q$ is inside $P$, or*
2. *there is a tower $t \in T(P)$ such that*
   (a) *$\overline{pq}$ intersects* base$(t)$*, or*
   (b) *$\overline{pq}$ passes below an edge of* cap$(t)$ *and above an edge of* base$(t)$*, or*
   (c) *$\overline{pq}$ passes below an edge of* cap$(t)$ *and $p$ or $q$ is above* base$(t)$*.*

Lemma 2 allows us to treat a segment-emptiness query as the disjunction of several different conditions and test separately for each of these conditions. Developing data structures for each of these condition is relatively routine; they can be implemented using standard multi-level range-searching data structures. Below we provide some of the details.

**Lemma 3** *Let $\mathcal{P}$ be a set of disjoint $\beta$-fat convex polyhedra in $\mathbb{R}^3$ of total complexity $n$. We can detect whether there is an endpoint of a query segment $s$ inside a polyhedron in $\mathcal{P}$ using a data structure that requires $O(n/\beta)$ storage and preprocessing time and has $O((1/\beta)\log n)$ query time.*

**Proof.** In order to detect whether a point $p$ is inside a polyhedron in $\mathcal{P}$, we use the so-called *object BAR-tree* designed by De Berg and Streppel [6]. This is a BSP tree with $O(n)$ nodes and depth $O(\log n)$, such that every leaf region intersects at most $O(1/\beta)$ objects. Therefore, assuming the polyhedra have constant complexity, we can test whether $p$ is inside any of the polyhedra in $\mathcal{P}$ simply by finding the cell containing $p$ in $O(\log n)$ time and then testing whether $p$ is inside any of the polyhedra in the cell. If the polyhedra do not have constant complexity, we apply the Dobkin-Kirkpatrick hierarchy [7] to each polyhedron. In either case, the test takes $O(\log n)$ to determine which cell $p$ is in and $O((1/\beta)\log n)$ to test if $p$ is inside any of the $O(1/\beta)$ polyhedra meeting that cell. □

**Lemma 4** *Assuming there is no endpoint of query segment $s$ inside any polyhedron in $\mathcal{P}$, we can detect whether $s$ intersects any polyhedron in $\mathcal{P}$ using a data structure which requires $O(n^{2+\varepsilon}/\beta^2)$ storage and preprocessing time and has query time $O((\log n)/\beta^2)$. Furthermore, for any $m$ with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon}/\beta^2)$ storage and preprocessing time and has $O((n/(\beta^2\sqrt{m}))\log n)$ query time.*

**Proof.** There are three cases to consider, according to Lemma 2. We will design a different structure for each of them, and in each case we will need a separate structure for each of the $|\mathcal{D}|$ canonical tower directions. Let us fix one of the canonical directions $\vec{d}$, and let $\mathcal{T} = \mathcal{T}(\vec{d})$ be the set of all towers of that direction. Without loss of generality, assume that the base of the towers in $\mathcal{T}$ is parallel to the $xy$-plane.

*Condition 2a: $s$ intersects* base$(t)$ *for some tower $t \in \mathcal{T}$:* A segment $s$ intersects base$(t)$ if and only if $s$ intersects base$(t)$ both in the projection onto the $yz$-plane and in the projection onto the $xz$-plane. Hence, we can test whether there is an intersected base using a four-level partition tree: the first two levels are used to select the bases that are intersected in the projection onto the $xz$-plane, and the last two levels are used to test whether any of these bases are also intersected in the projection onto the $yz$-plane.

Conditions 2b and 2c can be handled similarly. □

**Putting it all together.** By combining the data structure for segment-emptiness queries described above with Agarwal and Matoušek's parametric search technique [1] mentioned earlier, we obtain our final result.

**Theorem 5** *Let $\mathcal{P}$ be a set of $\beta$-fat convex and disjoint polyhedra in $\mathbb{R}^3$ of total complexity $n$. We can preprocess $\mathcal{P}$ using $O(n^{2+\varepsilon}/\beta^2)$ storage and preprocessing time, such that ray-shooting queries can be answered in $O((\log^2 n)/\beta^2)$ time. Moreover, for any $m$ with $n < m < n^2$, we can construct a structure that uses $O(m^{1+\varepsilon}/\beta^2)$ preprocessing time and storage such that queries take $O((n/\beta^2\sqrt{m})\log^2 n)$ time.*

## References

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[2] P. K. Agarwal and J. Matoušek. On range-searching with semi-algebraic sets. *Discrete and Computational Geometry*, 11:393–418, 1993.

[3] M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal.* Springer-Verlag New York, LNCS 703, 1993.

[4] M. de Berg. Vertical ray shooting for fat objects. In *Proc. 21st Annual Symposium on Computational Geometry*, pages 288–295, 2005.

[5] M. de Berg and C. Gray. Vertical ray shooting and computing depth orders for fat objects. In *Proc. 17th Annual Symposium on Discrete Algorithms*, 2006. To appear.

[6] M. de Berg and M. Streppel. Approximate range searching using binary space partitions. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 110–121, 2004.

[7] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241–253, 1983.

[8] M. J. Katz. 3-d vertical ray shooting and 2-d point enclosure, range searching, and arc shooting amidst convex fat objects. *Computational Geometry: Theory and Applications*, 8:299–316, 1997.

[9] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. *International Journal of Computational Geometry and Applications*, 7(4):317–347, 1997.

[10] S. Mohaban and M. Sharir. Ray shooting amidst spheres in three dimensions and related problems. *SIAM Journal on Computing*, 26(3):654–674, 1997.

[11] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.

[12] M. Pellegrini. Ray shooting and lines in space. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 599–614. CRC Press, Boca Raton-New York, 1997.

[13] M. Sharir and H. Shaul. Ray shooting and stone throwing. In *Proc. 11th European Symposium on Algorithms*, pages 470–481. Springer-Verlag, LNCS 2832, 2003.

[14] M. Sharir and H. Shaul. Ray shooting amid balls, farthest point from a line, and range emptiness queries. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 525–534, 2005.

[15] A. F. van der Stappen. *Motion Planning Amidst Fat Obstacles.* PhD thesis, Dept. of Computer Science, Utrecht University, 1994.

# Approximation of an open polygonal curve with a minimum number of circular arcs

R. L. Scot Drysdale[*] Günter Rote[†§] Astrid Sturm[‡§]

**Abstract**

An algorithm for approximating a given open polygonal curve with a minimum number of circular arcs is introduced. In computer-aided manufacturing environments the paths of cutting tools are usually described with circular arcs and straight line segments. Greedy algorithms for approximating a polygonal curve with curves of higher order can be found in the literature. Without theoretical bounds it is difficult to prove anything about the quality of these algorithms. We present an algorithm which allows us to build a directed graph of all possible arcs and look for the shortest path from the start point to the end point of the polygonal curve. We can prove a runtime of $O(n^2 \log n)$, for $n$ the number of vertices of the original polygonal chain.

## 1 Introduction

In computer-aided manufacturing enviroments tool paths are usully made of line segments and circular arcs [3, 4, 5]. Approximating data by curves of higher order [1, 2, 3, 4, 5, 6, 7, 8] has been investigated extensively in the past. The theoretical bounds of these problem are not as well studied.

We wish to approximate a polygonal chain $P = (p_1, \ldots, p_n)$ by a series of circular arcs (which could include straight lines, as circles of infinite radius). The endpoints of the arcs are vertices of $P$. We want our approximating curve to have distance at most $\varepsilon$ from $P$. As a first approximation to this problem, one can look at a region formed from strips of width $\varepsilon$ centered at the polygon edges. However, in the vicinity of sharp corners, this does not guarantee that the curve remains close to the given points. Figure 1 shows a circular piece of a hypothetic curve that can shortcut the bend at $p_4$ if it is only required to remain in the strips. (Also, it might overshoot the bend, as indicated in the vicinity of $p_6$, although this looks like a theoretical possibility only.) To avoid this, we introduce a *gate* at every vertex. The approximating curve is required to pass through all gates in succession, and the curves are not allowed to pass through a gate twice. This will guarantee that any curve into a point $p_i$ can be joined with any curve out of $p_i$ without danger of an intersection other than at $p_i$.

For our problem, we assume that we are given a polygonal "tolerance region" $R$ and a sequence of gates $g_1, g_2, \ldots, g_n$, which are segments through the points $p_i$. We will refer to endpoints of gates lying to the left of $P$ as we walk from $p_1$ to $p_n$ as left endpoints and the other endpoints as right endpoints. We require that the gates do not cross. We require that $R$ is a simple polygon passing through all gate endpoints, that $R$ does not intersect the interior of gates or cross the segments connecting corresponding endpoints of successive gates, and that the sections of $R$ connecting $g_i$ with $g_{i+1}$ do not cross the lines that extend $g_i$ and $g_{i+1}$. (The line extending a segment is the line that contains that segment.)

Ideally, the gate $g_i$ at vertex $p_i$ is a line segment of length $2\varepsilon$ centered at $p_i$ that bisects the angle $p_{i-1}p_ip_{i+1}$. For a convolved curve with sharp bends close together, we might have to shorten the gates and to reduce the $\varepsilon$-strip around the edges, as shown in the right part of Figure 1.

Modeling the curve approximation problem by an appropriate tolerance region with gates is a problem of its own, which we do not discuss here. In Figure 1, we have chosen to approximate the "ideal" circular boundary at the outer angle of each vertex by a single edge of $P$. One can use more edges to get a finer approximation, or one could also choose to approximate the circular arc from inside, to get a guaranteed upper distance bound of $\varepsilon$. Our time bounds assume that the size of $R$ is proportional to $n$.

**Definition 1 (proper gate stabbing)** *A curve stabs gates $g_i, \ldots, g_j$ properly, if and only if:*

- *the curve passes through gate $g_m \in \{g_i, \ldots, g_j\}$ from the side of $\overline{p_{m-1}p_m}$ to the side of $\overline{p_mp_{m+1}}$*
- *the curve passes through every gate only once (the stabbing curve may pass through an endpoint of the gate in addition to passing through it once)*

[*]Department of Computer Science, Dartmouth College, scot@cs.dartmouth.edu

[†]Department of Computer Science, Free University Berlin, rote@inf.fu-berlin.de.

[‡]Department of Computer Science, Free University Berlin, sturm@inf.fu-berlin.de

[§]Partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS – Algorithms for Complex Shapes)

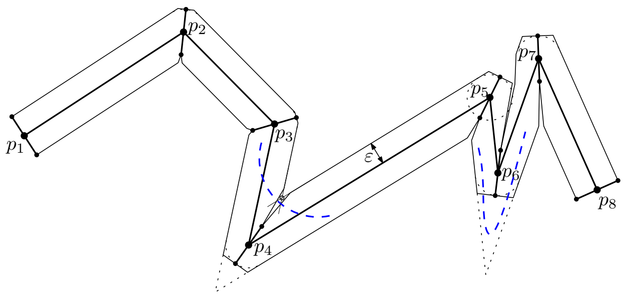


Figure 1: polygonal tolerance region

**Definition 2 (valid circular arc)** *A circular arc $a_{ij}$ with starting point $p_i$ and endpoint $p_j$ is a valid arc if:*

- *the arc stabs the gates $g_{i+1}, \ldots, g_{j-1}$ properly,*
- *the arc does not cross any piece of the boundary of the tolerance region $R$.*
- *the arc reaches $p_i$ from the correct side of $g_i$ and reaches $p_j$ from the correct side of $g_j$.*

Note that because $R$ passes through the gate endpoints, any arc that goes through a series of gates without crossing the tolerance boundary must go through them in the correct order, so we do not need to test for that separately.

We can split the problem of determining if a valid circular arc connects $p_i$ with $p_j$ into three parts. First, we compute all arcs between $p_i$ and $p_j$ that stab all intermediate gates properly. Second, we compute all arcs that start at $p_i$ and end at $p_j$, reaching both from the correct side. Third, we compute all arcs between $p_i$ and $p_j$ that do not intersect with the tolerance boundary. A valid circular arc has to be member of all three result sets.

## 2 Stabbing through the gates

**Definition 3 (point/gate bisectors)** *Given a point $p$ and a gate $g$, let $b_l$ be the bisector of $p$ and $g$'s left endpoint and $b_r$ be the bisector of $p$ and $g$'s right endpoint.*

**Lemma 1** *The centers of all circles passing through a vertex $p$ and intersecting a gate $g$ exactly once lie in a double cone whose boundary is $b_l$ and $b_r$. The sections we want are the ones where one half plane includes $p$ and the other excludes it. (Figure 2 illustrates this.) In the degenerate case where $b_l$ is parallel to $b_r$ one "cone" is the strip between the bisectors and the other "cone" is empty.*

**Proof.** A case analysis of circles centered on the bisectors and in each of the regions confirms the claim. □

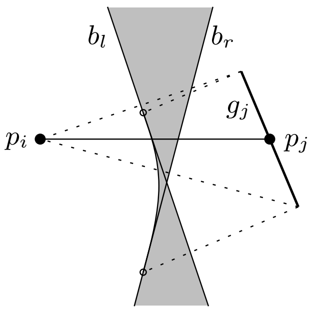


Figure 2: region of all centers of circles passing through $p_i$ and gate $g_j$

**Lemma 2** *The region of the centers of all circles passing through a vertex $p$ which are tangent to the gate $g$ or intersect it twice forms a parabolic region (in Figure 2 the parabolic region is the filled region to the left of the double cone). The boundary of the parabolic region is given by a parabolic piece, defined by the centers of the circles which are tangent to the gate, and by two pieces of the boundary double cone. In the degenerate case when the bisectors are parallel the parabolic region is empty.*

**Proof.** Geometric analysis proves the claim. □

The centers of all circles which pass through a point $p$ and intersect or are tangent to the gate are in the union of the double cone and the parabolic region described in Lemma 1 and Lemma 2. By Definition 1 an arc stabs the gates properly only if every gate is intersected only once. Circles centered in the parabolic region, intersect the gate twice or (on the parabola) never pass through the gate. Therefore we include only the two straight boundary segments of the parabolic region, which represent extreme cases of passing through a boundary point and an additional point on the gate. We exclude the rest of the parabolic region as centers for intermediate gates, even though in some circumstances when points on $P$ are relatively close together an arc might end at an endpoint on $P$ before it again intersects the gate. We will allow points in the parabolic region at endpoints of the arc.

According to Lemma 1, one cone is the region of the centers of disks including the left boundary point of the gate and excluding the right boundary point. Circular arcs centered in these region pass the gate from the correct side, according to the stabbing condition, if they are in CCW orientation. In CW orientation the arc would walk around the left boundary point before intersecting the gate. The unbounded part of this cone lies to the left of $P$. Symmetrically the circular arcs in the other cone need CW orientation to pass the gate in the correct direction, and the unbounded part of this cone lies to the right of $P$.

So from now on we talk about the left cone and the right cone. A circular arc stabbing through the gates is not allowed to change its orientation.

**Lemma 3** *A circular arc $a$ starting at a point $p$ stabs gates $g_i, \dots, g_j$ properly if and only if its center lies in the intersection of the left cones defined by $p$ and the gates, or the intersection of the right cones.*

**Proof.** Straightforward. □

**Lemma 4** *Incremetally computing the two regions of centers of all valid circular arcs passing through a point $p$ and stabbing all $g_i, \dots, g_j$ gates properly, can be done in $O(n \log n)$ time.*

**Proof.** It is the incremental intersection of $O(n)$ half planes. □

## 3 Arc endpoints

A valid circular arc from $p_i$ to $p_j$ must reach each point from the correct side of its gate. All arcs that start at $p_i$ and end at $p_j$ have their centers on the bisector of the segment connecting $p_i$ and $p_j$. However, some of these arcs will approach the gate from the wrong side. We want to eliminate all such arcs, but allow arcs that are tangent to a gate at its defining point or which would circle back and cut the gate a second time if the arc did not stop. This means that we want to consider not only centers in the double cone, but also in the parabolic region.

All points on the bisector of $p_i$ and $p_j$ can be centers of arcs from $p_i$ to $p_j$ that reach $p_j$ from the correct side of $g_j$, but most can only go around the circle in one direction. The exception is the circle which is tangent to $p_j$ at $g_j$. We call the center of this circle the splitting point. All points on the ray of the bisector that starts at the splitting point and goes right are centers of CW arcs that meet $p_j$ from the correct side. All points on the ray of the bisector that starts at the splitting point and goes left are valid centers of CCW arcs.

To determine which arcs meet $p_i$ from the correct side of $g_i$ we do the symmetrical test, with the roles of $p_i$ and $p_j$ reversed.

**Lemma 5** *Let $b$ be the perpendicular bisector of the segment between $p_i$ and $p_j$. Let $s_i$ be the point of $b$ which is the center of a circle tangent to $g_i$ at $p_i$, and let $s_j$ be defined symmetrically. The centers of all CW arcs that reach both $p_i$ and $p_j$ from the correct side are all points on $b$ to the right of both $s_i$ and $s_j$. CCW arcs are symmetrical.*

**Proof.** This ray is the intersection of the CW rays for both endpoints of the arc. □

## 4 Tolerance boundary

We break the tolerance boundary $R$ into two polygonal chains, one on each side of the original polygonal chain $P$. When dealing with CCW circles we will exclude the right chain, which we call the CCW boundary. When dealing with CW circles we will exclude the left chain, which we call the CW boundary. The requirements that the arcs pass through gates and that the tolerance boundary not intersect the interiors of gates or cross the segments connecting boundary points of successive gates guarantees that there will be no conflict with the other boundary.

A circle passing though point $p$ does not intersect or contain any edge on a polygonal chain $C$ if its center lies closer to $p$ than to any point on $C$. That is, if we compute the Voronoi diagram of $C \cup p$, the center of the circle must lie in point $p$'s region, $V(p)$.

This is not quite the condition that we want, namely that a circular arc does not cross chain $C$. The Voronoi region guarantees that an entire circle does not cross $C$. However, in our case these are equivalent.

**Lemma 6** *If an arc from $g_i$ to $g_j$ does not intersect a tolerance boundary between $g_i$ and $g_j$ then neither does the circle on which that arc lies.*

**Proof.** For each pair of consecutive gates $g_k$ and $g_{k+1}$ we are given that the section of the arc between them does not intersect the section of the tolerance boundary between $g_k$ and $g_{k+1}$. But this section of the tolerance boundary is not allowed to cross either the segment connecting its start and end points or the lines extending $g_k$ and $g_{k+1}$. Therefore this section of the boundary cannot intersect the rest of the circle, either. □

While we could compute the entire Voronoi diagram of $C \cup p$ to determine $V(p)$, this would be too expensive. Fortunately, we can interatively add $n$ consecutive segments of $C$ and update $p$'s Voronoi region $V(p)$ in $O(n)$ total time.

Voronoi regions are "generalized star shaped". This means that a shortest segment from a boundary point to a nearest point in the shape defining the region lies entirely within the region.

**Lemma 7** *Each segment added will either cause no change to $V(p)$ or will replace a section of $V(p)$ by at most three new segments (two straight lines and a parabola). (If $V(p)$ is unbounded we think of an edge "at infinity" connecting the two infinite rays, so that these three "segments" are considered consecutive.)*

**Proof.** Follows from the connectedness of $C$ and the generalized star-shaped property. □

There are two parts to updating $p$'s Voronoi region when adding a segment $s$ to the diagram. First, we find a place on the boundary of $V(p)$ that is equidistant from $p$ and $S$, if such a place exists. If so, we next walk around the boundary of $V(p)$, eliminating boundary sections until we reach the other place on the boundary where $p$ is equidistant from $S$. (Note that either of these places could be "at infinity".)

The second part is easy - walk around the boundary of $V(p)$ from the starting point, eliminating obsolete bisector segments until you get to the finish point.

Because $C$ is a polygonal chain, the first part is also easy. $V(p)$ is bounded by bisector pieces between $p$ and a subset of the segments in $C$. Of the segments in this subset, there is a first segment $F$ and a last segment $L$, according to the order along the chain.

**Lemma 8** *If $V(p)$ changes, then its boundary with either $V(F)$ or $V(L)$ must change.*

**Proof.** The proof formalizes the idea if you can't go through the chain $C$, then the only way to get to $V(p)$ is through $V(F)$ or $V(L)$. □

**Lemma 9** *We can compute the centers of all circular arcs that pass between $g_i$ and $g_j$ without crossing the tolerance boundary in $O(n)$ time.*

**Proof.** Incrementally add segments from $C$ and amortize the update time. □

## 5 Computing the shortest path

To determine the shortest path from the start point to the end point of the polygonal curve we can build a directed graph of all possible valid arcs and do a BFS to find the shortest path from $p_1$ to $p_n$.

**Theorem 10** *A point $c$ is the center of a valid CW circular arc from $p_i$ to $p_j$ if and only if it is in the intersection of:*

- *the intersection of the right cones between $p_i$ and each of the gates $g_{i+1}$ through $g_{j-1}$.*
- *The region of $V(p_i)$ in the Voronoi diagram of $p_i$ and all of the segments on the CW boundary between $g_i$ and $g_j$.*
- *all points on $b$ to the right of both $s_i$ and $s_j$, where $b$, $s_i$, and $s_j$ are as defined in Lemma 5.*

*The conditions for valid CCW arcs are symmetrical.*

**Proof.** Direct consequence of earlier lemmas. □

We find the possible arcs from a point $p_i$ to all points further along $P$ incrementally. We maintain the intersection of the right cones, the intersection of the left cones, the Voronoi region of $p_i$ with the CW boundary, and the Voronoi region of $p_i$ with the CCW boundary. At each step we update each of the four items. We intersect each bisector ray with an intersection of cones and with a Voronoi region, and then test if the two intersections overlap.

Note that we can quit if both of the cone intersection regions become empty. In fact, we could quit when the intersection of the right cones with the CW boundary Voronoi region and the intersection of the left cones with the CCW boundary Voronoi region are both empty, if we could test this quickly.

**Theorem 11** *Given an open polygonal curve $P = (p_1, \ldots, p_n)$, a polygonal tolerance boundary of size $O(n)$, and a gate for each $p_i$, we can approximate $P$ by a minimum number of valid circular arcs in $O(n^2 \log n)$ time.*

**Proof.** Each starting point takes $O(n \log n)$ time. □

## 6 Future Work

Because we compute all possible circular arcs from $p_i$ to $p_j$, we expect to be able to use this information to match tangents of successive arcs or to compute bi-arcs. We have partial results along these lines.

## References

[1] J. Eibel. Approximation of Planar Curves Within an Asymmetric Tolerance Band. *MSc.thesis, Universität Salzburg, Computerwissenschaften*, 2002.

[2] M. Held and J. Eibel. Biarc Approximation of Polygons Within Asymmetric Tolerance Bands. *Computer-Aided Design*, 37:357–371, 2005.

[3] DS. Meek and DJ. Walton. Approximation of discrete data by $G^1$ arc splines. *Computer-Aided Design*, 24:301–306, 1992.

[4] DS. Meek and DJ. Walton. Approximating quadratic NURBS curves by arc splines. *Computer-Aided Design*, 25:371–376, 1993.

[5] DS. Meek and DJ. Walton. Approximating smooth planar curves by arc splines. *Journal of Computational and Applied Mathematics*, 59:221–231, 1995.

[6] L. Piegl. Curve Fitting for Rough Cutting. *Computer-Aided Design*, 18:79–82, 1986.

[7] J. Schönherr. Smooth Biarc Curves. *Computer-Aided Design*, 25:365–370, 1993.

[8] M. Yeung and DJ. Walton. Curve Fitting With Arc Splines for NC Toolpath Generation. *Computer-Aided Design*, 26:845–849, 1994.

# How to Sample and Reconstruct Curves With Unusual Features

Tobias Lenz
Institut für Informatik, Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany
tlenz@mi.fu-berlin.de

## Abstract

This work generalizes the ideas in the Nearest-Neighbor-Crust algorithm by Dey and Kumar. It allows to reconstruct smooth closed curves from $\varepsilon$-samples with $\varepsilon \leq 0.48$ which is a big improvement compared to the original bound. Further generalization leads to a new algorithm which reconstructs arbitrary curves (open, closed, smooth, with corners, with intersections) in any dimension. The algorithm works well in practice but lacks a nice sampling condition comparable to the well-known $\varepsilon$-sampling condition. This shortcoming is posed as an open problem.

## 1 Introduction

In curve reconstruction, one cannot or does not want to transfer a complex shaped curve. Instead, a finite *sample* of the original curve is generated which must become reconstructed at the destination. A reconstruction is just a polygonal line connecting the points in the correct order. The approximation quality measured as distance to the original can be arbitrarily bad depending on the sample. It is obvious, that not every point set leads to the correct reconstruction, therefore the sample must have special properties called the *sampling condition*.

## 2 Picking Samples from a Curve

**Definition 1** *A* sample *$S$ is a finite subset of a curve $\Sigma$. The elements of the sample are points in $\mathbb{R}^d$ and are called* sample points.

The $\varepsilon$-sampling condition basically states that the sample should be dense if the curvature is high or parts of the curve are close together while it might be loose in straight regions of the curve. Formally it is defined as follows.

**Definition 2** *The* medial axis *of a smooth curve is the set $M$ containing the center points of all empty circles which touch the curve in more than one point.*

*The* local feature size *of a point $p \in \Sigma$ is defined as* $lfs(p) = \min_{m \in M} \|p - m\|$.

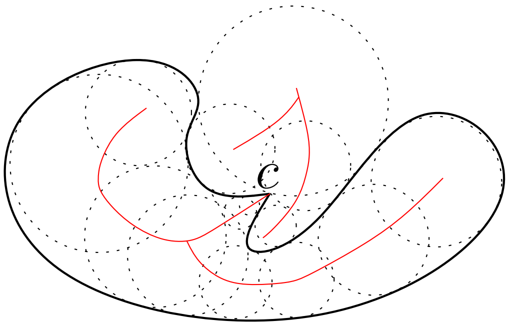


**Figure 1:** A non-smooth curve (bold, black) with some medial balls (dotted) and the medial axis (red) intersecting the curve at the corner point $c$.

**Definition 3** *A sample $S$ is an* $\varepsilon$-sample *of the smooth curve $\Sigma$ if and only if $\forall p \in \Sigma : \exists s \in S : \|p - s\| \leq \varepsilon \cdot lfs(p)$.*

These definitions apply well to smooth curves but an $\varepsilon$-sample is impossible for curves with corners or intersections because the medial axis goes through these points and hence the local feature size becomes zero. Therefore the sampling density should be infinite. See figure 1 for an illustration. A possible solution is to exclude a small neighborhood of these points from the $\varepsilon$-sampling condition and define a new one for these regions. This is used for corners in [2].

## 3 The NN-Crust Algorithm

The basis for this work came from Dey and Kumar and their *Nearest-Neighbor-Crust* algorithm [1]. They start with a point set and connect each point $p$ to its nearest neighbor $u$ and its nearest half-neighbor $h$. Consider the line perpendicular to $\overline{pu}$ through $p$ which splits the plane. The half-neighbor $h$ of $p$ is now the point closest to $p$ which does not lie in the half plane containing $u$.

This procedure results in a set of edges forming the reconstruction. If the underlying point set is at least a $\frac{1}{3}$-sample of a smooth closed curve, the NN-Crust algorithm guarantees a correct reconstruction. Dey and Kumar showed that the edges in the reconstruction are a subset of the edges of a Delaunay triangulation of the point set. This allows simple implementations

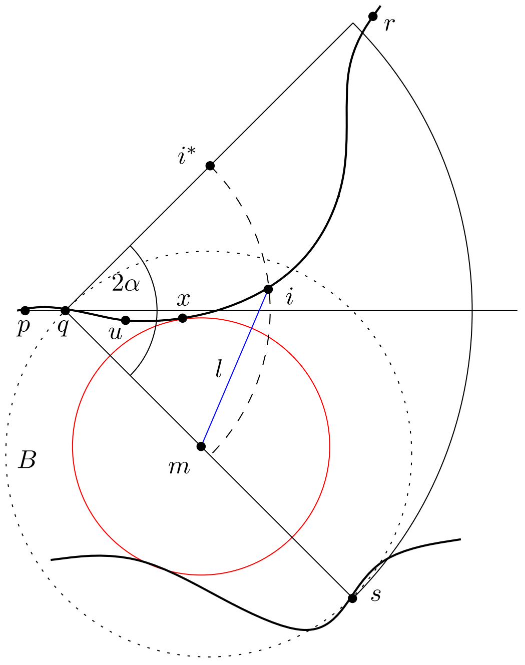


**Figure 2:** Computing the worst case distance between a point on the curve and the medial axis.

with $\mathcal{O}(n \log n)$ running time in 2d and extensions to higher dimensions.

## 4 Generalization of the NN-Crust Algorithm

### 4.1 Flexible Opening Angles

Changing the way of describing the half-neighbor slightly allows a simple generalization. A similar definition to the one given in section 3 is the following: The half-neighbor of a point $p$ with nearest neighbor $u$ in a point set $S$ is the point $h \in S \setminus \{p\}$ which minimizes the distance between $p$ and $h$ and fulfills $\angle \overline{up}, \overline{ph} > \frac{\pi}{2}$. Now the angle appears as a parameter and instead of $\frac{\pi}{2}$ one can choose an arbitrary angle. These points are no longer called "half-neighbors" but *$\alpha$-neighbors* instead, where $\alpha$ specifies the maximally allowed turning angle going from $u$ to $p$ to $h$.

The remainder of this section is used to show that for $\alpha$-neighbors with $0 < \alpha \leq \frac{\pi}{2}$, the resulting edges are still a correct reconstruction. The arguments refine and extend ideas from the original paper which leads to a much better bound for $\varepsilon$. The proofs are skipped due to space constraints.

**Lemma 1** *The angle spanned by three adjacent samples in an $\varepsilon$-sample with $\varepsilon < 1$ is at least $\pi - 4 \arcsin \frac{\varepsilon}{2}$. This angle corresponds to $\pi - \alpha$, which gives $\varepsilon \leq 2 \sin \frac{\alpha}{4}$ if $\alpha$ is fixed and less than $\frac{\pi}{2}$.*

Figure 2 shows how to bound $\varepsilon$ from below using the following arguments.

**Lemma 2** *Given three adjacent sample points $p, q, r$ in an $\varepsilon$-sample with $\varepsilon < 1$, the curve segment between $q$ and $r$ runs completely inside the cone at $q$ aligned with $(p, q)$ with opening angle $2\alpha = 8 \arcsin \frac{\varepsilon}{2}$.*

**Lemma 3** *Consider a $\frac{1}{2}$-sample. Given a chain of three adjacent samples $p, q, r$ and a sample $s$ not adjacent to $q$. If $p$ and $r$ do not lie inside the ball $B$ with diameter $\overline{qs}$ there will be a medial axis point on the segment $\overline{qs}$ with distance at most $\frac{3}{4}\|s - q\|$ from $q$.*

**Lemma 4** *Given an $\varepsilon$-sample of a smooth closed curve with $\varepsilon \leq 0.48$ and a correct edge $(p, q)$. The edge $(q, s)$ is correct if and only if $s$ is the closest point to $q$ inside the cone with apex $q$ aligned to $(p, q)$ with opening angle $2 \cdot 0.97$.*

Altogether this guarantees correct results for a whole family of algorithms with different angles $0 < \alpha \leq \frac{\pi}{2}$ and for $\varepsilon \leq 0.48$ in the extreme case which is a big improvement compared to the NN-Crust [1]. The NN-Crust algorithm is a special case of this approach for $\alpha = \frac{\pi}{2}$. It follows a direct increase of the $\varepsilon$ bound for the original algorithm from $\frac{1}{3}$ to 0.4 due to a more careful analysis.

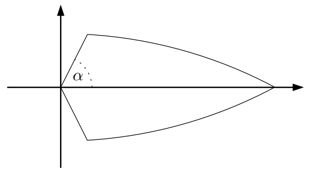
### 4.2 Non-Circular Neighborhoods

Picking the nearest neighbor essentially is like growing a circle around a point until another point touches the circle's boundary. Instead of growing circles one can use other shapes, here called *probes*, for example with the intention to prefer straight segments over bends or left turns over right turns. This requires an alignment of the shape because it is no longer rotationally symmetric. Since every vertex in a reconstruction of a smooth closed curve has degree two, one only has to find a single *seed edge* and grow the reconstruction from that, aligning the shape at the tip of an already reconstructed edge. A good choice for the seed edge is the overall shortest edge which is obviously a correct edge for $\varepsilon$-samples.

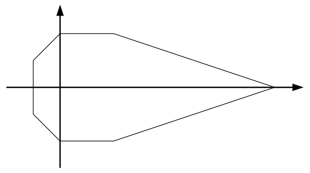
Compared to the global approach of triangulating a point set and then selecting a subset of the triangulation edges, tracing is a more local concept. Extending the reconstructed graph at its loose ends with minimum weight components is also a very natural procedure for greedy algorithms. This idea is in line with famous algorithms like Dijkstra's shortest path algorithm or Prim's algorithm to construct a minimum spanning tree.

An animated demonstration and an interactive version to experiment with can be found in [3].

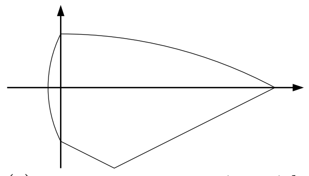
**Definition 4** *A continuous map $\theta : [-\alpha; \alpha] \to \mathbb{R}^+$ is called $\alpha$-probe for $0 < \alpha \leq \pi$ if and only if $\beta > \gamma \geq 0 \Rightarrow \theta(\beta) \leq \theta(\gamma)$ and $\beta < \gamma \leq 0 \Rightarrow \theta(\beta) \leq \theta(\gamma)$.*



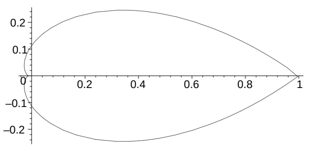
(a) $\alpha$-probe, symmetric, without negative extend



(b) polygonal, convex, symmetric, with negative extend



(c) convex, asymmetric, with negative extend



(d) $\theta(\alpha) = 4^{-\|\alpha\|}$

**Figure 3:** Four probes illustrating the variety of possible shapes. All these probes can be described with constant complexity. Probe (a) is assembled from two circular arcs, (b) is a polygon with seven vertices, (c) is a combination of circular arcs and straight segments and (d) is an exponential functions of the angle.

*For $\alpha = \pi$, $\theta(\pi) = \theta(-\pi)$ must be true additionally.*

*An $\alpha$-probe $\theta$ is* symmetric *if and only if $\forall \beta \in (0; \alpha] : \theta(\beta) = \theta(-\beta)$.*

*An $\alpha$-probe $\theta$ has* negative extend *if and only if $\alpha > \frac{\pi}{2}$.*

The name "probe" refers to the shape one obtains from drawing the function $\theta$ in polar coordinates with $\theta$ as distance from the origin.

Inflating a probe is equivalent to minimizing the following distance function.

**Definition 5** *Let $\theta$ be an $\alpha$-probe by definition 4 and $\beta = \angle \overline{pq}, \overline{qr}$. The* probe distance function *for three points $p, q, r$ is defined as*

$$F_{pq}(r) = \begin{cases} \infty & \text{if } \|\alpha\| < \|\beta\| \\ \frac{\|q-r\|}{\theta(\beta)} & \text{otherwise.} \end{cases}$$

Please note that $F_{pq}$ is strictly positive and directed and in general different from $F_{qp}$.

This definition is valid for any dimension since only the plane spanned by $p, q, r$ is considered and $F_{pq}(r)$ is computed in that plane.

The proof from section 4.1 still applies for arbitrary $\alpha$-probes with $\alpha \leq \frac{\pi}{2}$ by changing the distance $\|q-r\|$ to $\min\limits_{-\alpha \leq \beta \leq \alpha} (\theta(\beta))$ and $\|q - s\|$ to $\max\limits_{-\alpha \leq \beta \leq \alpha} (\theta(\beta))$.

## 5 Practical Extensions

### 5.1 Corners and Endpoints

Sharp corners might become reconstructed with probes with negative extend, first going straight into the corner and then backwards out to a close point (if the apex of the corner is in the sample). Obviously corners with an apex angle of $\beta$ can only be reconstructed with $\alpha$-probes with $\alpha \geq \pi - \beta$. Endpoints are the extreme case of corners where the point one came from is the point with minimum distance, i.e. $F_{pq}(p)$ is minimal.

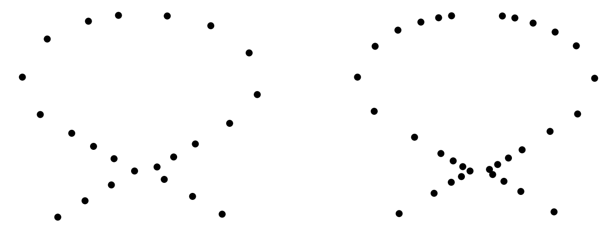


**Figure 4:** Higher density perceptually indicates disconnectedness. The sample on the left shows nearly equally distributed sample points while the sample on the right uses higher density close to endpoints and corners to emphasize the upcoming gap.

One idea to realize this comes from human perception and should be clear from figure 4. Changing the local density of the sample makes intended gaps much more noticeable. By placing a single additional sample point $p$ very close to an endpoint $q$, a $\pi$-probe aligned at the edge $(p, q)$ with a tiny extend in the backward direction will detect $p$ as the $\pi$-neighbor of $q$. This edge is already part of the recontruction, so the algorithm stops this branch with a vertex of degree one.

Unfortunately an $\varepsilon$-sample is now no longer possible because the medial axis goes through corners, see figure 1. It is also no longer sufficient to guarantee a correct reconstruction from an $\varepsilon$-sample because the distance between two adjacent samples must not be too small if none of them is an endpoint. The latter problem can be solved by switching to $(\varepsilon, \delta)$-samples which have the additional condition that for any two samples $p, q \in S$, $\|p - q\| > \delta \cdot \mathrm{lfs}(p)$. This approach is not investigated here, because the necessity for infinite sampling density at corners and intersections remains even for $(\varepsilon, \delta)$-samples.

### 5.2 Intersections

An observation for smooth curves or smooth parts of curves in the vicinity of intersection points is that the angle between segments of properly placed sample points is small while this angle often abruptly increases connecting to a wrong point. Therefore a key idea is to favor small angles over small distances and hope to "tunnel through" a narrow set of wrong points and reach the correct one. This can be realized easily using probes.

A sample as described throughout section 5 always exists. Some construction hints follow but a nice and general rule seems to be hard to find.

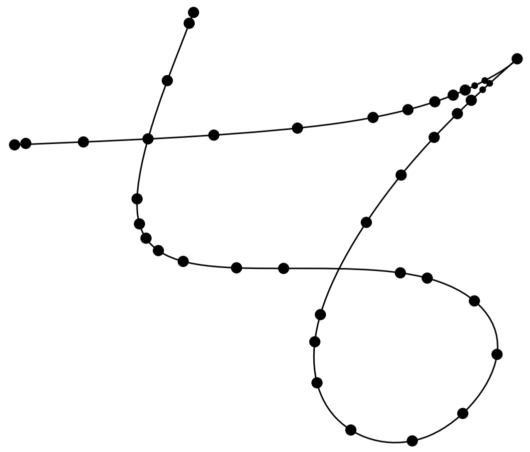


**Figure 5:** A curve with endpoints, a corner and two intersections. The sample points show the rule of thumb how to sample these features.

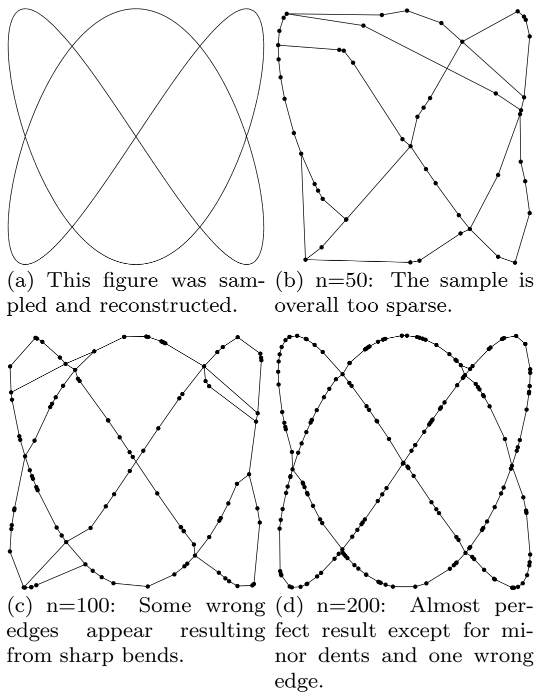
For the smooth parts of the curve, an $\varepsilon$-$\delta$-sample suffices. Corners with apex angle $\beta$ can be sampled loosely like smooth bends without the apex in the sample. If the apex is part of the sample, the distance to the closest point on both edges is given by the negative extend of the used probe. If an intersection point is in the sample, the reconstruction of its vicinity should be simple. Otherwise the adjacent samples should all have roughly the same distance to the intersection point. In the end, one could find the closest pair of sample points and add an even closer point to each endpoint to guarantee correct reconstruction of open curves. Figure 5 gives an intuition of the described rules of thumb.

## 6 Experimental Results

The Lissajou figure in figure 6(a), given by the parametric form $L(t) \mapsto (\sin 4\pi t, \cos 6\pi t)$, was sampled such that $n$ values were taken equally distributed from $[0; 1)$ and the corresponding points were put into the sample. This was done for several values of $n$.

The results depend of course on the random input but they show a general behavior which can be reproduced for other inputs of the same size. One possible drawback of the algorithm is that a single failure—a single wrong edge—can lead to a chain reaction for the following edges because they all base on a wrong edge. The experiments show that this disastrous effect is not likely to occur if the sample has at least a certain density.

Obviously one can always create a point pattern around an intersection which will result in a wrong reconstruction independent of the density. Nevertheless there are no additional wrong edges besides close to intersections and no gaps, so the reconstructed topology will be correct if the sample is dense enough. This



(a) This figure was sampled and reconstructed. (b) n=50: The sample is overall too sparse. (c) n=100: Some wrong edges appear resulting from sharp bends. (d) n=200: Almost perfect result except for minor dents and one wrong edge.

**Figure 6:** Experimental results from random samples.

suggests that the algorithm can also be used successfully in a heuristic approach.

## 7 Open Problem

The major open problem is the lack of a nice sampling condition. The $\varepsilon$-sampling condition is well-established far beyond the problem of curve reconstruction but it is just not applicable for corners and intersections. Excluding regions around these artifacts and handling them with special cases is possible but definitively not a very nice solution.

Is there a sampling condition, as simple and clear as the $\varepsilon$-sampling condition, which also holds for non-smooth or even self-intersecting curves?

## References

[1] T. K. Dey and P. Kumar. A simple provable algorithm for curve reconstruction. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 893–894, Jan. 1999.

[2] S. Funke and E. A. Ramos. Reconstructing a collection of curves with corners and endpoints. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 344–353. Society for Industrial and Applied Mathematics, 2001.

[3] T. Lenz. Reconstructing collections of arbitrary curves. In *SCG '05: Proceedings of the 21st annual symposium on Computational geometry*, pages 366–367. `http://compgeom.poly.edu/acmvideos/socg05video/index.html`, 2005.

# On the Curve Equipartition Problem: a brief exposition of basic issues

Costas Panagiotakis * George Georgakopoulos * George Tziritas *

## Abstract

We describe briefly the problem of partitioning a continuous curve into N parts with equal chords. (The length of a chord may be defined by any smooth distance metric applied on its endpoints-the Euclidean metric being one of them.) A have proved that a decision variation of this problem is NP-complete, yet for any continuous curve and any N there always exists at least one equipartition. In this work, we propose an approximate algorithm and also a steepest descent method that converges to an exact solution.

## 1 Introduction

The curve segmentation problem is a challenging problem of computational geometry. A huge number of applications, like object recognition and tracking, signal summarization and compression, curve simplification and computer graphics applications, are based on curve segmentation. Many computer graphics applications are based on curve segmentation problem, like surface simplification and 3D modelling. Most polygonal surface simplification methods employ triangles as their approximating elements when constructing a surface [3]. One of the most popular triangulation methods is Delaunay triangulation [5], [11].

On computer vision applications the curve segmentation problem also appears. Signal summarization and key frames detection methods using an appropriate feature set reduce the initial problem into a curve segmentation problem [4]. Methods for non articulated motion tracking are based on solutions of the curve equipartition problem [10].

Another example of such segmentation approach is the 2D or 3D polygonal approximation [2] or convex polygons [7]. This problem asks for computing another polygonal curve that approximates the original curve. The problem can be formulated in two ways [6], [1] : The problem of minimum error $(Min-\varepsilon)$ and the the problem of minimum number of line segments $(Min - \#)$.

| Symbols | Definitions |
|---|---|
| $C(t)$ | The given curve, $t \in [0, 1]$ |
| $N$ | The number of equal length chords |
| $d(x, y)$ | $d(x, y) = \lvert C(x) - C(y)\rvert_2$ |
| $U_{M\cdots K}$ | $\{M, M + 1, \cdots , K\}$ |

Table 1: Proof symbol table.

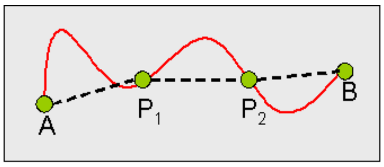


**Fig. 1:** An EP example for $N = 3$, $|AP_1| = |P_1P_2| = |P_2B|$.

### 1.1 Problem Definition

The equipartition problem is defined as follows: Let $C(t), t \in [0, 1]$ be a 2D acyclic curve[1] that starts on $A = C(0)$ and ends on $B = C(1)$. We have to compute $N - 1$ sequential curve points $P_i, i \in U_{1\cdots N-1}$, $P_0 = A$, $P_N = B$ under the constraint $d(P_{i-1}, P_i) = d(P_i, P_{i+1})$, $i \in U_{1\cdots N-1}$. The problem is the curve partitioning into $N$ parts with equal chords, so that the first starts from $A$ and the last ends on $B$ (Fig. 1). Some useful symbols are defined on Table 1.

The solution is obvious for $N = 1$, as we have one chord, $AB$. When $N = 2$, we have to locate a curve point $P_1$, so that $|AP_1| = |P_1B|$. This point can be given as the intersection of the curve with the $AB$ segment bisector. When $N$ is higher than two, there is not a trivial method to compute the equal length chords. The above problem can have more than one solutions depending on curve shape and the value of $N$. As $N$ tends to infinity the problem solution (equal length chords) will be unique and it will approximate the curve (Fig. 2). More examples can be found in [9]. By our analysis, the EP problem admits always a solution, thus the decision version of EP is certainly not NP-complete. Yet, at the time, we cannot give

*Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, Greece {cpanag, ggeo, tziritas}@csd.uoc.gr

[1]We suppose that the curve is piecewise-algebraic. The problem can be defined in the same way in any dimension $(C(t) \in \Re^n)$.

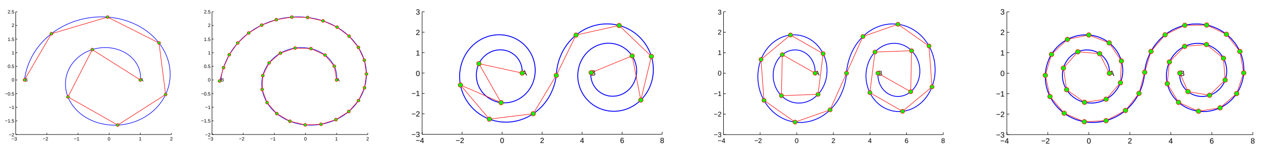


**Fig. 2:** EP examples for different N. The higher the $N$, the better the curve approximation.

a guarantee that it admits always a succint solution, thus we cannot assert that the functional version of it belongs to the TFNP class. Finally consider the following 'verification' version of the EP: given a curve C and a distance $\rho$ is it possible to equipartition $C$ into $N$ parts, so their $N$ chords are all equal to $\rho$? For this version, we do know something positive: we can prove that this version of EP is NP-complete (by a reduction from Knapsack).

### 1.2 An Equivalent Definition of the Problem

The smooth function $d(x, y) = |C(x) - C(y)|_2,\ \ x, y \in [0, 1]$ gives a different view and an equivalent definition of the problem. We can use as $d(x, y)$ any smooth metric like Euclidean distance. As a smooth metric distance, $d(x, y)$ is characterized by the following properties:

1. $d(x, y) = 0 \Leftrightarrow x = y$ (isolation).
2. $d(x, y) = d(y, x)$ (symmetry).
3. $d(x, y)$ inherits continuity existence from $C(t)$.
4. $d(x, y)$ can be defined in any dimension ($C(t) \in \Re^n$).

An equivalent problem definition can be derived using $d(x, y)$. A problem solution $\{0, t_1, t_2, \cdots, t_{N-1}, 1\}$ of curve $C(t)$, corresponds to the surface $d(x, y)$ as a point sequence, $(0, t_1), (t_1, t_2), \cdots, (t_{N-1}, 1)$. The length $r$ of each chord is given by the following equation:

$$r = d(0, t_1) = d(t_1, t_2) = \cdots = d(t_{N-1}, 1) \qquad (1)$$

An alternative problem definition will be the determination of $\{t_1, t_2, \cdots, t_{N-1}\}$, so that Equation (1) will be satisfied. Under this definition we prove that the problem has at least one solution.

The rest of the paper is organized as follows: A brief description of the proof of solution existence for each chord number is presented in Section 2. The proposed algorithms that solve the problem are presented in Sections 3 and 4. Conclusions and discussion are provided in Section 5.

## 2 Existence Proof

In this section we are going to give a brief description of the proof that there exists at least one solution for each $N$. We are going to analyze the case of $N = 3$. The cases of $N > 3$ are faced with a generalization of the used methodology for $N = 3$ and their proof can be done inductively. The proof is presented with more details in [8].

The function $f_2(x, y) = d(x, y) - d(x, 0),\ x \in [0, 1], y \geq x$, is continuous and partially monotonous function. The null plane curves[2] of $f_2(x, y)$ will be continuous and partially monotonous. The total solutions of equal length chords for $N = 2$ are given by the points $(x, y)$ of these curves, because $d(x, 0) = d(x, y)$ and $y \geq x$. Let $h_2(s) = [a_2(s), b_2(s)], s \in [0, 1]$ be the curve of $f_2(x, y)$ null plane, that starts from $[0, 0]$ ($h_2(0) = [0, 0]$). Then $a_2(s) \leq b_2(s)$, because the points of $h_2(s)$ are points of $f_2$ domain. This curve exists as $f_2(0, 0) = 0$. It can be proved that $h_2(s)$ ends on $y = 1$, equivalently $b_2(1) = 1$. We consider the continuous function $q(s)$ (Equation (2)). Using this function, we will find $\{t_1, t_2\}$ with $t_2 \geq t_1$ satisfying Equation (1) and the proposition will have been proved for $N = 3$.

$$q(s) = d(a_2(s), b_2(s)) - d(1, b_2(s)), \quad s \in [0, 1] \qquad (2)$$

It holds that,

- $q(0) = d(0, 0) - d(1, 0) = -d(1, 0) < 0$ and
- $q(1) = d(a_2(s), 1) - d(1, 1) = d(a_2(s), 1) > 0$

At least a $s_2 \in (0, 1)$ exists (applying the Bolzano theorem) so that $q(s_2) = 0$. This means $d(a_2(s_2), b_2(s_2)) = d(1, b_2(s_2))$. Let $t_2 = b_2(s_2)$ and $t_1 = a_2(s_2)$, $\Rightarrow t_2 \geq t_1$ and $d(t_1, t_2) = d(t_2, 1)$. The $(t_1, t_2)$ is a point of $h_2 \Rightarrow d(t_1, t_2) = d(0, t_1)$. Thus we have found $\{t_1, t_2\}$ with $t_2 \geq t_1$ satisfying the equation (1). Finally, the problem has been proved for $N = 3$.

## 3 Iso-Level Algorithm (ILA)

The iso-level algorithm is based on the equivalent problem definition. It computes at least one solution

[2]The null plane curves of $f(x, y)$ are defined by the equation $f(x, y) = 0$.

or all the solutions (greedy version). It is inductive. Thus, when it is executed for $N$, it solves the problem for any number of parts (with equal chords) less than $N$.

The major hypothesis of the method is that the function $d(x, y), x, y \in [0, 1]$ can be approximated by a polygonal surface $\hat{d}(x, y)$. Thus, the $\hat{d}(x, y)$ is determined by $d(m_k, m_l), k, l \in \{1, 2, \cdots, M\}$. Let

$$D_{ij} = [x_i, x_{i+1}] \times [y_j, y_{j+1}] \subset [0, 1]^2$$

with $x_i = y_i = m_i$, $i, j \in \{1, 2, \cdots, M\}$. The segment $D_{ij}$ can be separated into two triangles: $D^1_{ij}$ where $x - x_i \geq y - y_j$ and $D^2_{ij}$ where $x - x_i < y - y_j$. Under our major hypothesis, we have considered that $\hat{d}(x, y), x, y \in D^1_{ij}$ or $x, y \in D^2_{ij}$ is a part of plane.

In each iteration step $l$, the algorithm computes the curves $L_l$ so that if the point $(u, v) \in L_{l-1}, u > v$, then, it holds that $(z, u), z > u \in L_l \Leftrightarrow d(u, v) = d(z, u)$. These curves consist of line segments defined on $D^1_{ij}$, $D^2_{ij}$, so they can be computed from the line segments end points. For $l = 1$, it holds that,

$$L_1 = [(0, 0), (m_1, 0)] \cup \cdots \cup [(m_{M-1}, 0), (1, 0)].$$

Let $(x, y) \in L_l$, $x > y$. Under the above definition, the equipartition of curve $C(t), t \in [0, x]$ into $l$ chords can be done using the precomputed curves $L_l, L_{l-1}, \cdots, L_1$ (see Fig. 3). The equipartition of curve $C(t)$, $t \in [0, 1]$ into $l + 1$ chords can be done using the curves $L_l, L_{l-1}, \cdots, L_1$. Let $q_l(u, v) = d(u, v) - d(u, 1)$, $(u, v) \in L_l$, $u > v$. This function is piecewise linear. The roots of this function will give the last two points $(\dot{t}_{l-1}, \dot{t}_l)$ of the equipartition. The other points are estimated using the rule of Fig. 3.

It can be proved that for each step there is a continuous curve $h_l \subset L_l$ starting from $[0, 0]$ and ending on axis $x = 1$ or $y = 1$ (see Fig. 3). We can compute at least one solution of the problem using these curves. The computation cost of $h_l$ curves is $O(M \cdot N)$, because we can track them starting from their known end point $[0, 0]$. We can estimate a normalized error $(NE)$ of an estimated equipartition of length chords by getting the standard deviation of the estimated length chords of this equipartition divided by the mean length chord of this equipartition ([8]). $NE$ is decreased as $M$ increases. It can be proved that $NE$ is decreased by the factor $O(\frac{1}{M^2})$.

Figure 4 illustrates the results of this proposed algorithm for different curves and values of $N$. The null plane curves converge to the diagonal $(y = x)$, as $N$ increases (see Fig. 4(e)), and there exist exactly one solution. At least one solution belongs on the $h_N(s)$ null plane curve. However, in some cases, more solutions appear on other null plane curves (see Fig. 4(a), 4(c)).

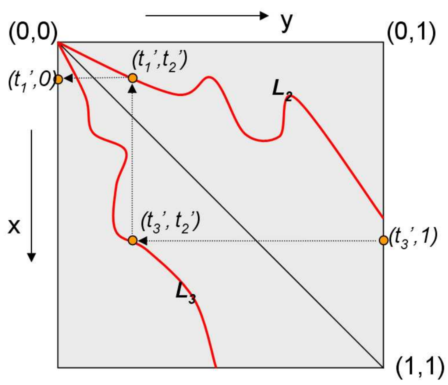


**Fig. 3:** An example of curve equipartition into 4 chords. It is shown the recursive computation of $\{\dot{t}_1, \dot{t}_2, \dot{t}_3\}$ and $L_2, L_3$ curves.

## 4 Steepest Descent based Method

The steepest descent based method (SDM) converges to the closest solution to an initial equipartition, given this initial equipartition. The major advantage of this method is that the computed chords will have exactly the same length, as the end of the last chord is converging to $B$. In some cases the algorithm can not converge as there may appear local minima or jumps (loops) between different solutions. These phenomena are increased, when the initialization is far enough from an existing solution. But, when the problem has a unique solution, which is usually observed for high $N$, then the algorithm will converge. A pseudocode of this procedure is given hereafter.

**Steepest Descent based Algorithm**

```
$s = r_0$
$P_0 = A$
Repeat
    for i=1:N
        $P_i = C(t_i) : (t_i > t_{i-1}) \wedge (|P_{i-1}P_i| = s)$
    end
    $s = \begin{cases} s + \lambda\frac{|B-P_N|}{N}, & P_N \in \text{inside of curve } C(t) \\ s - \lambda\frac{|B-P_N|}{N}, & P_N \in \text{ouside of curve } C(t) \end{cases}$
Until $|P_N - B| < T$
```

The learning rate $\lambda$ determines the number of steps which are needed for convergence. However, when $\lambda$ is set to a high value ($\lambda > 0.5$), it can cause instability and not convergence. For better convergence, we can start the method with $\lambda \approx 0.5$ and we decrease it to $\lambda \approx 0.05$. Conclusively, if we want to solve the problem for lower $N$, where there are possibly many solutions and local minima, it is better to execute first the approximate algorithm, getting a good initialization for the steepest descent based algorithm. For higher $N$, where the problem might have a unique solution, the proposed method will converge

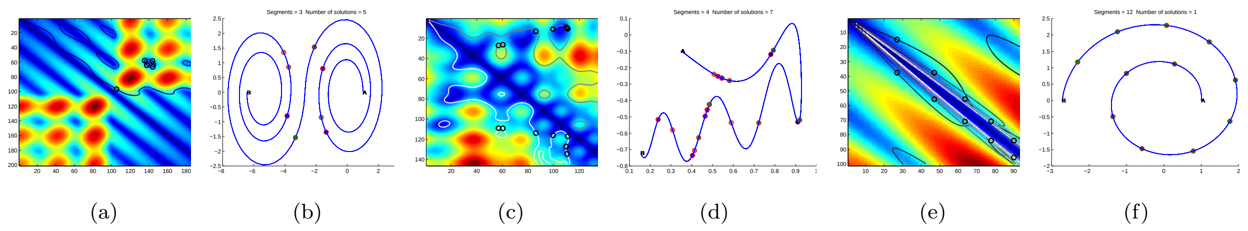


**Fig. 4:** Results of greedy version of ILA. The estimated solutions are projected on $d(x, y)$ (left) with black cycles and on input curve $C(t)$ (right) with the same color points belonging to the same equipartition. The null plane curves are projected on $d(x, y)$, with gray colors, at both sides of diagonal $x = y$.

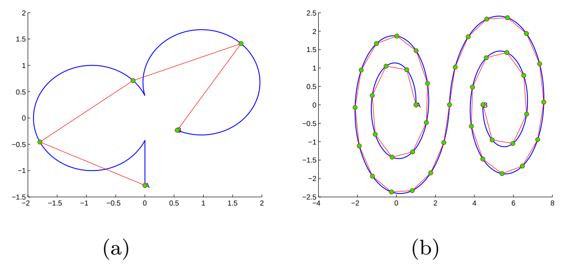


**Fig. 5:** Steepest descent based algorithm results.

towards the exact solution, even if the initialization is not close to the solution. The time complexity of the algorithm is $O(N \cdot S)$, where $S$ denotes the number of steps that are needed for convergence. $S$ depends on curve shape and how close to the final solution the initialization is. Results of the steepest descent based algorithm are shown in Fig. 5.

## 5 Conclusions

In this paper, we have discussed the curve equipartition problem (EP). If the chord length is part of the instance then the problem becomes NP-complete, but if the chord-length can be chosen at will it can be proved that there always exists at least one solution for any piecewise linear curve and thus for any continuous curve. An equivalent definition of the problem (using a distance metric on $[0..1] \times [0..1]$) leads also to an existence proof and to an approximate algorithm. The ILA can compute at least one solution or all the solutions using a greedy version of the algorithm. The output of this algorithm can be used to initialize a steepest descent based algorithm that converges to an exact solution.

## References

[1] P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23(2):273–291, 2000.

[2] K.-L. Chung, W.-M. Yan, and W.-Y. Chen. Efficient algorithms for 3-d polygonal approximation based on lise criterion. *Pattern Recognition*, 35:2539–2548, 2002.

[3] H. Delingette. General object reconstruction based on simplex meshes. *International Journal of Computer Vision*, 32:111–142, 1999.

[4] A. D. Doulamis, N. Doulamis, and S. Kollias. Non-sequential video content representation using temporal variation of feature vectors. *IEEE Trans. on Consumer Electronics*, 46:758–768, 2000.

[5] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *SIGGRAPH*, 1997.

[6] Y. Kurozumi and W. Davis. Polygonal approximation by the minimax method. *Computer Vision, Graphics and Image Processing*, pages 248–264, 1982.

[7] M. A. Lopez and S. Reisner. Hausdorff approximation of convex polygons. *Computational Geometry*, 32:139–158, 2005.

[8] C. Panagiotakis, G. Georgakopoulos, and G. Tziritas. The curve equipartition problem. *submitted to Computational Geometry, url = http://www.csd.uoc.gr/ cpanag/papers/EP.pdf*, 2005.

[9] C. Panagiotakis, G. Georgakopoulos, and G. Tziritas. Curve segmentation into equal segments. Technical Report CSD-TR-05-02, Computer Science Department, University of Crete, 2005 (in greek).

[10] C. Panagiotakis and G. Tziritas. Construction of animal models and motion synthesis in 3D virtual environments using image sequences. In *Proc. of the second Int. Symp. on 3DPVT*, 2004.

[11] C. Sohler. Fast reconstruction of delaunay triangulations. *Computational Geometry*, 31:166–178, 2005.

# Understanding the Inverse Ackermann Function

Raimund Seidel

Fachrichtung Informatik, Saarland University
Postfach 15 11 50, D-66041, Saarbrücken, Germany
rseidel@cs.uni-sb.de

The so-called inverse Ackermann function is an exceedingly slowly growing function that arises in bounds for a number of computational/combinatorial problems. The proofs of those bounds are usually tedious, proceed in a bottom-up fashion, and provide little (if any) intuition in the nature of the bound.

In my talk I will describe a top-down approach that is rather simple and leads naturally to such inverse Ackermann function bounds (without ever having to talk about the Ackermann function itself). The most striking example will be so-called path compression, which arises in algorithms for the union-find problem. This is one of the basic problems in algorithmics. It is distinguished by the fact that it is typically the only basic problem in intermediate level algorithms courses for which fast solutions are taught but not analyzed.

# On the density of iterated line segment intersections[*]

Ansgar Grüne[‡] Sanaz Kamali Sarvestani[‡]

**Abstract**

Given $S_1$, a finite set of points in the plane, we define a sequence of point sets $S_i$ as follows: With $S_i$ already determined, let $L_i$ be the set of all the line segments connecting pairs of points of $\bigcup_{j=1}^{i} S_j$, and let $S_{i+1}$ be the set of intersection points of those line segments in $L_i$, which cross but do not overlap. We show that with the exception of some starting configurations the set of all crossing points $\bigcup_{i=1}^{\infty} S_i$ is dense in a particular subset of the plane with nonempty interior. This region can be described by a simple definition.

## 1 Introduction

Given $S = S_1$, a finite set of points in the Euclidean plane, let $L_1$ denote the set of line segments connecting pairs of points from $S_1$. Next, let $S_2$ be the set of all the intersection points of those line segments in $L_1$ which do not overlap. We continue to define sets of line segments $L_i$ and point sets $S_i$ inductively by

$$\begin{aligned} L_i &:= \left\{ pq \;\middle|\; p, q \in \bigcup_{j=1}^{i} S_j \;\wedge\; p \neq q \right\}, \\ S_{i+1} &:= \{ x \mid \{x\} = l \cap l' \text{ where } l, l' \in L_i \}. \end{aligned}$$

Finally, let $S_\infty := \bigcup_{i=1}^{\infty} S_i$ denote the limit set.

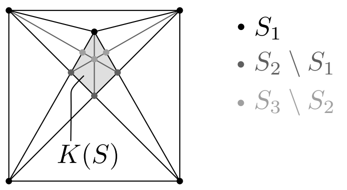


Figure 1: The first iterations $S_1$, $S_2$ and $S_3$ of line segment intersections and the candidate $K(S)$.

In this article we show in which region $K(S)$ the crossing points $S_\infty$ are dense, and in which exceptional cases the crossing points are not dense in any set with non-empty interior.

Several results concerning the density of similar iterated constructions are known, see Bezdek and Pach [2], Kazdan [10], Bárány, Frankl, and Maehara [1]. Ismailescu and Radoičić [8] examined a question very similar to ours. The only difference is that they considered lines instead of line segments. They proved by applying nice elementary methods that with the exception of two cases the crossing points are dense in the whole plane. Hillar and Rhea [7] independently proved the same statement with different methods.

Our setting of line segment intersections turns out to be more difficult. It has more exceptional cases, where the crossing points are not dense in any set with non-empty interior. And in non-exceptional cases the crossing points are not dense in the whole plane but only in a particular convex region $K(S)$. In fact, if we do not have an exceptional configuration, the density of the line intersections in the whole plane is an easy consequence of the result presented here. This can be shown by the arguments displayed in Figure 4 of [8].

Our work is also motivated by another interesting problem introduced recently by Ebbers-Baumann et al. [4], namely how to embed a given finite point set into a graph of small dilation. For a given geometric graph $G$ in the plane and for any two vertices $p$ and $q$ we define their *vertex-to-vertex dilation* as $\delta_G(p,q) := |\pi(p,q)|/|pq|$, where $\pi(p,q)$ is a shortest path from $p$ to $q$ in $G$ and $|\cdot|$ denotes the Euclidean length. The *dilation of* $G$, $\delta(G)$, is the maximum dilation of any two vertices.

A geometric graph $G$ of smallest possible dilation $\delta(G) = 1$ is called *dilation-free.* We will give a list of all cases of dilation-free graphs in the plane in Section 2. Given a point set $S$ in the plane, the *dilation of* $S$ is defined by $\Delta(S) := \inf\{\, \delta(G) \mid G = (V,E) \text{ planar graph}, S \subset V \}$. Determining $\Delta(S)$ seems to be very difficult. The answer is even unknown if $S$ is a set of five points placed evenly on a circle. However, Ebbers-Baumann et al. [4] were able to prove $\Delta(S) \leq 1.1247$ for every finite point set $S \subset \mathbb{R}^2$ and they showed lower bounds for some special cases.

A natural idea for embedding $S$ in a graph of small dilation is to try to find a geometric graph $G = (V,E)$, $S \subseteq V$, such that $\delta_G(p,q) = 1$ for every $p,q \in V$. Now, suppose we have found such $G$. Obviously, for every pair $p,q \in S$, the line segment $pq$ must be a part of $G$. Since $G$ must be planar, every intersection point of these line segments must also be in $V$ and so on.

[*]The full version of this article is available as technical report [5] via internet. Parts of the results stem from Sanaz Kamali Sarvestani's master's thesis [9].

[‡]Department of Computer Science I, University of Bonn, {gruene,kamali}@cs.uni-bonn.de

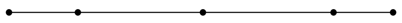
If this iteration produces only finitely many intersection points, i.e. the set $S_\infty$ defined at the very beginning of this article is finite, we have a planar graph $G = (V, E)$ with $S \subset V$, $V \setminus S$ finite and $\delta(G) = 1$ thus $\Delta(S) = 1$. This shows that $|S_\infty| < \infty$ can only hold if $S$ is a subset of the vertices of a dilation-free graph. We call those point sets *exceptional configurations.* Note that $\Delta(S) = 1$ could still hold for other sets. There could be a sequence of proper geometric graphs whose dilation does not equal 1 but converges to 1.

Our main result, Theorem 8, shows that in all the cases where $S$ is not an exceptional configuration, $S_\infty$ is dense in a region with non-empty interior. Very recently Klein and Kutz [11] proved a special case of this theorem and used it to prove the first non-trivial lower bound $\Delta(S) \geq 1.0000047$ which holds for every non-exceptional finite point set $S$.

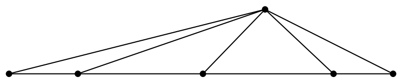
## 2 Exceptional Configurations and the Candidate

Here, we list all cases of dilation-free graphs. They can also be found at Eppstein's Geometry Junkyard [6]. It can be proven by case analysis that these are all possibilities. The *exceptional configurations* are the subsets of the vertices of such graphs.

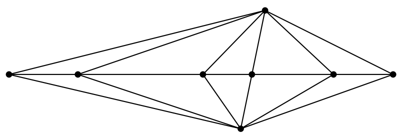
$(i)$ $n$ points on a line



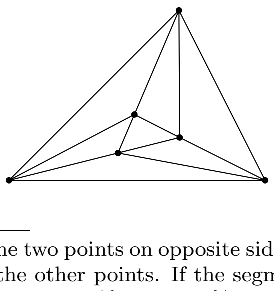
$(ii)$ $n-1$ points on a line, one point not on this line



$(iii)$ $n-2$ points on a line, two points on opposite sides of this line[1]



$(iv)$ a triangle (i.e. three points) nested in the interior of another triangle. Every pair of two inner points is collinear with one outer point.



Next, we define the region $K(S)$, cf. Figure 1. Until we prove that $S_\infty$ is dense in $K(S)$, we call $K(S)$ the candidate.

**Definition 1** *The* candidate $K(S)$ *is defined as the intersection of those closed half planes which contain all the starting points except for at most one point:*

$$K(S) := \bigcap_{p \in S} \bigcap_{H \supset S \setminus \{p\}} H$$

*The second intersection is taken over all closed half planes $H$ which contain $S \setminus \{p\}$.*

It is not difficult to prove that the candidate is a convex polygon whose vertices belong to $S_1 \cup S_2$, and that every intersection point lies inside of the candidate, that is $S_\infty \setminus S \subset K(S)$, see the full paper [5].

## 3 Density in a Triangle

**Lemma 1** *(Main Lemma) Let the starting configuration $S = \{A, B, C, D, E\}$ be as follows: 3 points are the vertices of the triangle $\triangle ABC$, another 2 points $D$ and $E$ are on different sides of this triangle (see Figure 2), then $S_\infty$ is dense in a triangle.*

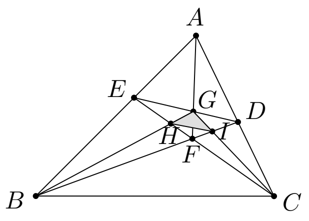


Figure 2: $S_\infty$ is dense in $\triangle GHI$.

**Proof.** We use arguments from projective geometry; see Bourbaki [3] for an introduction. Consider the Euclidean plane $\mathcal{A}$ as embedded in the projective plane $\mathbb{P}^2$. The complement of the projective line through $BC$ in $\mathbb{P}^2$ is an Euclidean plane $\mathcal{A}'$. On $\mathcal{A}'$ we have the same points as in $\mathcal{A}$ like shown in Figure 3.

We use this simple topological fact: A set $A$ is dense in a set $B$ with respect to a topological subspace $Y \subset X$ where $A \subset B \subset Y$ iff $A$ is dense in $B$ with respect to the whole space $X$. That means to prove, that $S_\infty$ is dense in a triangle in the Euclidean plane $\mathcal{A}$, it is sufficient to show, that $S_\infty$ is dense in the same triangle in the projective plane and it is also equivalent to prove, that $S_\infty$ is dense in that triangle with respect to $\mathcal{A}'$. In this case we want ro prove

[1]Let $p_1$ and $p_2$ be the two points on opposite sides of the line, and let $p_3, \ldots, p_n$ be the other points. If the segment $p_1p_2$ intersects with the convex hull $\mathrm{ch}(\{p_3, \ldots, p_n\})$, the intersection point must be a vertex of the dilation-free graph. However, it does not have to be part of the corresponding exceptional configuration.

that $S_\infty$ is dense in $\triangle GHI$ with $G := ED \cap AF$, $H := BG \cap EF$ and $I := CG \cap FD$.

In $\mathcal{A}'$ we have $AD \parallel EF$, because the lines through $AD$ and $EF$ in $\mathcal{A}$ cross each other in $C$ and $AE \parallel DF$, because the lines through them cross each other in $B$ respectively. Hence $G$ is the midpoint of $DE$, $I$ is the midpoint of $DF$ and $H$ is the midpoint of $EF$. But for this special case we can prove with elementary methods that the intersection points are dense in $\triangle GHI$ (see Section 3 of the full paper [5]). Hence $S_\infty$ is dense in $\triangle GHI$ in $\mathcal{A}$. □

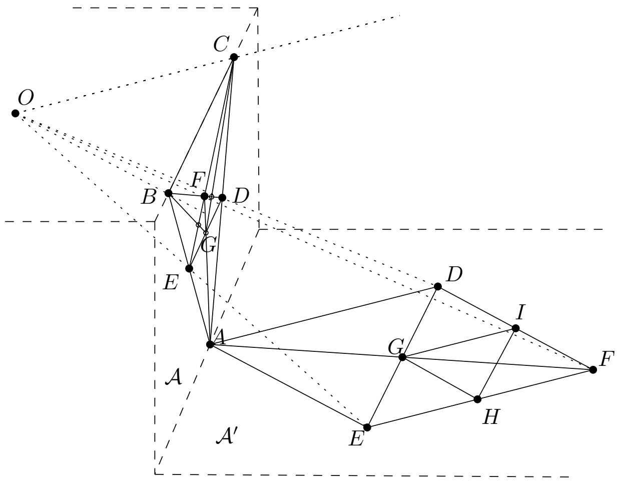


Figure 3: A useful projection of $\triangle ABC$

This implies the first partial result, proved as Corollary 7 in the full version [5]:

**Corollary 2** *Let $S_1$ be a set of $n > 4$ points in convex position in the plane no three of them on a line. Let $S_i$ be the sets defined as above, then $S_\infty = \bigcup_{i=1}^{\infty} S_i$ is dense in a triangle.*

We generalize this statement as follows.

**Lemma 3** *For any non exceptional configuration, there exists a triangle, in which $S_\infty$ is dense.*

This can be proved by detailed case analysis and by applying the main lemma, Lemma 1 (see proof of Lemma 8 in [5]).

## 4 Density in the Candidate

We mention the following two techical tools without proof (see the proofs in Section 5 of [5]).

**Lemma 4** *Let $L$, $M$ and $N$ be three distinct points such that $M$ lies on the line segment $LN$ and let $a$ and $b$ be two rays, emanating from $M$ on the same side of the line through $L$ and $N$. Let $K_0$ be a point on the ray $a$ (cf. Figure 4). We define a sequence of points as follows: $P_i := b \cap NK_i$, $K_{i+1} := LP_i \cap a$. Then the point sequence $(P_i)_{i\in\mathbb{N}}$ converges to $M$ on $b$ and the point sequence $(K_i)_{i\in\mathbb{N}}$ converges to $M$ on $a$.*

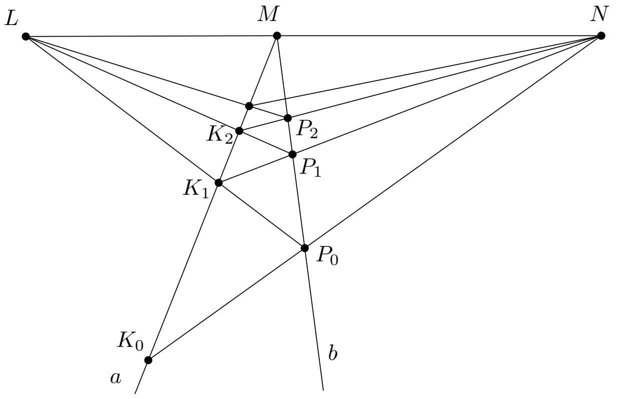


Figure 4: The point sequences $(P_i)_{i\in\mathbb{N}}$ and $(K_i)_{i\in\mathbb{N}}$ converge to $M$.

**Corollary 5** *Consider Figure 5: Let $L$ and $N$ be two points, and this time let $M$ be a point not on the line $LN$ but such that the line which goes through $M$ and is perpendicular to $LN$ meets the line segment $LN$. Let $H$ be the half plane bounded by the line which passes $LM$ and which does not contain $N$ and similarly let $H'$ be the half plane, bounded by the line which passes $MN$ and which does not contain $L$. Let $a$ and $b$ be two rays emanating from $M$ and lying in $H \cap H'$ and not on the boundary. If $K_0$ is a point on the ray $a$ and we define point sequences $(P_i)_{i\in\mathbb{N}}$ and $(K_i)_{i\in\mathbb{N}}$ analogously to Lemma 4, then these sequences converge to $M$.*

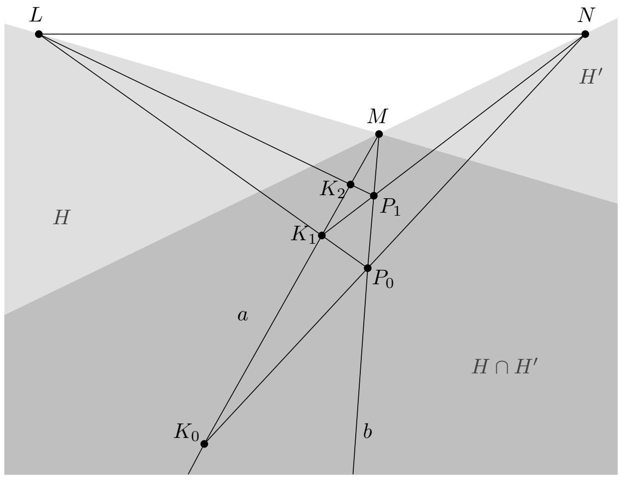


Figure 5: The point sequences $(P_i)_{i\in\mathbb{N}}$ and $(K_i)_{i\in\mathbb{N}}$ converge to $M$.

**Definition 2** *Let $\mathcal{C}$ be a convex polygon and let $P$ be a point outside of $\mathcal{C}$. We call the two rays emanating from $P$, which touch the boundary of $\mathcal{C}$,* tangents *of $\mathcal{C}$ through $P$. Let $A$ and $B$ be those two vertices of $\mathcal{C}$, which are the first to be touched by the tangents. We define the* visibility cone *of $P$ with respect to $\mathcal{C}$, $V(P,\mathcal{C})$, as follows:*

$$V(P,\mathcal{C}) := \triangle ABP - \mathcal{C}$$

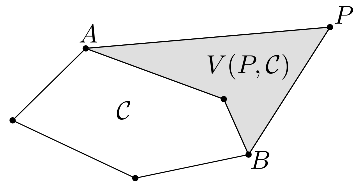


Figure 6: The visibility cone of $P$ with respect to $\mathcal{C}$, $V(P,\mathcal{C})$

In the following we often use this simple fact (see the proof in Attachment A of [5]):

**Fact 6** *A family of dense rays emanating from a fixed point generates dense intersection points on every line segment which is hit by the rays.*

**Lemma 7** *Let the starting configuration $S$ be an arbitrary finite point set. Assume that $S_\infty$ is dense in a convex region $\mathcal{R} \subset \mathrm{ch}(S)$ with nonempty interior. And let $P$ be a point from $S_\infty$ such that $P \notin \mathcal{R}$ and $P$ is not a vertex of $\mathrm{ch}(S)$. Then $S_\infty$ is dense in $\mathrm{ch}(\{P\} \cup \mathcal{R})$.*

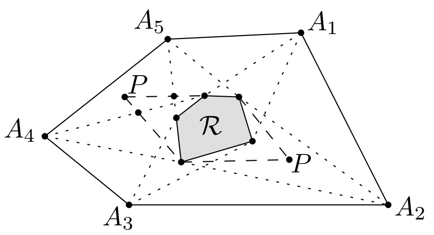


Figure 7: Two cases in the proof of Lemma 7.

**Proof.** Let $A_i$, $i = 1 \dots n$ be the vertices of $\mathrm{ch}(S)$.

We distinguish the following cases:

**Case 1** : If the point $P$ lies in or on the boundary of a visibility cone $V(A_i,\mathcal{R})$ for some $i$ say $i_0$, then we have $V(P,\mathcal{R}) \subset V(A_{i_0},\mathcal{R})$. Thus by Fact 6, $S_\infty$ is dense in $V(P,\mathcal{R})$ and therefore in $\mathrm{ch}(\{P\} \cup \mathcal{R})$.

**Case 2** : Else: $V(P,\mathcal{R})$ intersects at least one $V(A_i,\mathcal{R})$.

Hence $S_\infty$ is dense in the intersection of these visibility regions. In particular $S_\infty$ is dense in a sub line segment of at least one tangent of $\mathcal{R}$ emanating from $P$. Now we can apply Corollary 5 or Lemma 4 combined with Fact 6 ($P$ as $M$, the tangents as the half lines $a$ and $b$, and the neighbor side of the convex hull as $LM$, cf. Figure 7). ☐

**Theorem 8** *Let $S = S_1$ be a set of $n$ points in the plane, which is not an exceptional configuration. Then $S_\infty$ is dense in the candidate $K(S)$.*

**Proof.** If $S$ is not an exceptional configuration, by Lemma 3 we know that $S_\infty$ is dense in a triangle $T$. Furthermore we have seen in Section 2 that $T \subset K(S)$ and that every vertex of $K(S)$ is an intersection point. Let $P_1, P_2, \dots, P_n$ be the vertices of $K(S)$. Then using Lemma 7 inductively yields that $S_\infty$ is dense in $\mathrm{ch}(\{P_1, P_2, \dots, P_n\}) = K(S)$. ☐

## References

[1] I. Báránay, P. Frankl, and H. Maehara. Reflecting a triangle in the plane. *Graphs and Combinatorics*, 9:97–104, 1993.

[2] K. Bezdek and J. Pach. A point set everywhere dense in the plane. *Elemente der Mathematik*, 40(4):81–84, 1985.

[3] N. Bourbaki. *General Topology, part 2*, chapter VI, §3, pages 44–53. Elements of Mathematics. Hermann, Paris, 1966.

[4] A. Ebbers-Baumann, A. Grüne, M. Karpinski, R. Klein, C. Knauer, and A. Lingas. Embedding point sets into plane graphs of small dilation. In *Algorithms and Computation: 16th International Symposium, ISAAC 2005*, volume 3827 of *Lecture Notes Comput. Sci.*, pages 5–16. Springer, December 2005.

[5] A. Grüne and S. Kamali Sarvestani. On the density of iterated line segment intersections. Technical Report 004, Department of Computer Science I, University of Bonn, 2005. *http://www.cs.uni-bonn.de/I/publications/gk-disi-05.pdf.*

[6] D. Eppstein. The geometry junkyard: dilation-free planar graphs. Web page, 1997. *http://www.ics.uci.edu/˜eppstein/junkyard/dilation-free/.*

[7] C. Hillar and D. Rhea. A result about the density of iterated line intersections in the plane. *Comput. Geom. Theory Appl.*, to appear.

[8] D. Ismailescu and R. Radoičić. A dense planar point set from iterated line intersections. *Comput. Geom. Theory Appl.*, 27(3):257–267, 2004.

[9] S. Kamali Sarvestani. On the density of iterated segment intersections. Master's thesis, University of Bonn, 2005.

[10] D. A. Každan. Uniform distribution in the plane. *Transactions of the Moscow Mathematical Society*, 14:325–332, 1965.

[11] R. Klein and M. Kutz. The density of iterated crossing points and a gap result for triangulations of finite point sets. Unpublished manuscript, 2005.

# On the structure of sets attaining the rectilinear crossing number [*]

Oswin Aichholzer[†] David Orden[‡] Pedro A. Ramos[§]

## Abstract

We study the structural properties of the point configurations attaining the rectilinear crossing number $\overline{cr}(K_n)$, that is, those $n$-point sets that minimize the number of crossings over all possible straight-edge embeddings of $K_n$ in the plane. As a main result we prove the conjecture that such sets always have a triangular convex hull.

The techniques developed allow us to show a similar result for the halving-edge problem: For any $n$ there exists a set of $n$ points with triangular convex hull that maximizes the number of halving edges. Moreover, we provide a simpler proof of the following result from [13]: any set of points in the plane in general position has at least $3\binom{j+2}{2}$ $(\leq j)$-edges. This bound is known to be tight for $0 \leq j \leq \lfloor \frac{n}{3} \rfloor - 1$. In addition, we show that for point sets achieving this bound the $\lfloor \frac{n+3}{6} \rfloor$ outermost convex layers are triangles.

## 1 Introduction

Given a graph $G$, its *crossing number* is the minimum number of edge crossings over all possible drawings of $G$ in the plane. Crossing number problems have both, a long history, and several applications to discrete geometry and computer science. We refrain from discussing crossing number problems in its generality, but instead refer the interested reader to the early works of Tutte [15] or Erdös and Guy [8], the recent survey by Pach and Tóth [14], or the extensive online bibliography by Vrt'o [16].

In 1960 Guy [10] started the search for the *rectilinear crossing number* of the complete graph, $\overline{cr}(K_n)$, which considers only straight-edge drawings. The study of $\overline{cr}(K_n)$ is commonly agreed to be a difficult task and has attracted a lot of interest in recent years, see e.g. [1, 2, 3, 6, 7, 13]. In particular exact values of $\overline{cr}(K_n)$ are only known up to $n = 17$, see [4], and also the exact asymptotic behavior is still unknown. Several relations to other structures, like for example $k$-sets, are conjectured [11], but surprisingly little is known about the combinatorial properties of optimal sets.

Therefore in this paper we consider structural properties of point sets minimizing the number of crossings, that is, attaining the rectilinear crossing number $\overline{cr}(K_n)$. Relations are obtained by using basic techniques, like e.g. motion flips and rotational sweeps.

Moreover we draw connections to $j$-edges and $k$-sets which provide additional insight for two other prominent problems (see [5, 13]): Maximizing the number of halving edges and counting $(\leq k)$-edges. Let us recall that a *$j$-edge*, $0 \leq j \leq \lfloor \frac{n-2}{2} \rfloor$, is a segment spanned by the points $p, q \in S$ such that precisely $j$ points of $S$ lie in one open half space defined by the line through $p$ and $q$. In other words a $j$-edge splits $S \setminus \{p, q\}$ into two subsets of cardinality $j$ and $n-2-j$, respectively. Note that we consider non-oriented $j$-edges, i.e., the edge $pq$ equals the edge $qp$. A *$(\leq j)$-edge* has at most $j$ points in this half space, that is, it is a $k$-edge for some $0 \leq k \leq j$.

Due to the lack of space, we will omit proofs in this abstract.

## 2 Minimizing the number of rectilinear crossings

### 2.1 Order type flip events

Let $S = \{p_1, ..., p_n\}$ be a set of $n$ points in the plane in general position, that is, no three points lie on a common line. It is well known that crossing properties of edges spanned by points from $S$ are exactly reflected by the order type of $S$, introduced by Goodman and Pollack in 1983 [9]. The *order type* of $S$ is a mapping that assigns to each ordered triple $i, j, k$ in $\{1, ..., n\}$ the orientation (either clockwise or counter-clockwise) of the point triple $p_i, p_j, p_k$.

Consider a point $p_1 \in S$ and move it along a straight line in the plane in a continuous way. A change in the order type of $S$ occurs if, and only if, the orientation of a triple of points of $S$ is reversed during this process. This is the case precisely if $p_1$ is moved accross a line spanned by two other points, say $p_2$

[*]Research on this paper was carried out while the first author was a visiting professor at the Departamento de Matemáticas, Universidad de Alcalá, Spain.

[†]Institute for Software Technology, Graz University of Technology, oaich@ist.tugraz.at. Partially supported by the FWF (Austrian Fonds zur Förderung der Wissenschaftlichen Forschung) under grant S09205-N12, FSP Industrial Geometry

[‡]Departamento de Matemáticas, Universidad de Alcalá, david.orden@uah.es. Partially supported by grants MTM2005-08618-C02-02 and S-0505/DPI/000235.

[§]Departamento de Matemáticas, Universidad de Alcalá, pedro.ramos@uah.es. Partially supported by grants TIC2003-08933-C02-01, BMF2002-04402-C02-01 and S-0505/DPI/000235.

and $p_3$, of $S$. We call this event a *flip*. The three-dimensional analogous of these order type changes have been used in the study of the halving-edge problem in [5], where they were called mutations.

Assume that at time $t_0$ the three points $p_1, p_2, p_3$ are collinear, then the orientation of that triple at time $t_0 + \epsilon$ is inverse to its orientation at time $t_0 - \epsilon$ for an arbitrarily small constant $\epsilon > 0$. Let us assume that $p_1$ moves over the line segment $p_2p_3$ as indicated in Figure 1; otherwise we can interchange the role of $p_1$ and $p_2$ (or $p_3$, respectively) for the time interval $[t_0 - \epsilon, t_0 + \epsilon]$. We say that $p_1$ plays the *center role* of the flip. Note that for the last assumption we make use of the fact that no three points of $S$ are collinear, except for the moment when a flip is performed. Therefore we further assume that we do not stop the movement of $p_1$ during the flip, that is, in a collinear position.

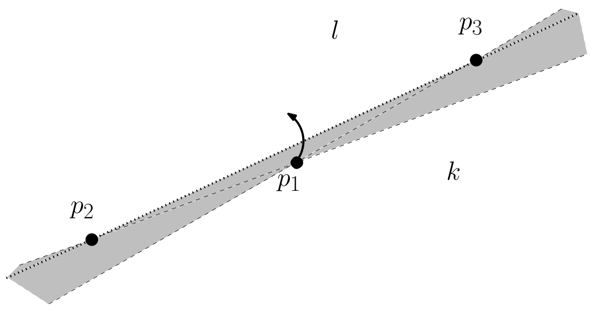


Figure 1: The point $p_1$ is flipped over the segment $p_2p_3$, changing the orientation of the triple $p_1, p_2, p_3$.

We call the above defined flip a $(k,l)$*-flip* if there are $k$ points on the same side of the line through $p_2$ and $p_3$ as $p_1$, excluding $p_1$, and $l$ points on the opposite side. Note that $k+l = n-3$. Our first goal is to study how flips affect the number of crossings of $S$, that is, the number of crossings of a straight-line embedding of $K_n$ on $S$, which will be denoted by $\overline{cr}(S)$. Note that we are only considering rectilinear crossings.

**Lemma 1** *A $(k,l)$-flip increases the number of crossings of $S$ by $k - l$.*

## 2.2 Halving rays

Since we know how flips affect the number of crossings, we are now interested in good moving directions. A point $p \in S$ is called *extreme* if it is a vertex of the convex hull of $S$. Two extreme points $p, q \in S$ are called non-consecutive if they do not share a common edge of the convex hull of $S$. We define a *halving ray* $\ell$ to be an oriented line passing through one extreme point $p \in S$, avoiding $S \setminus \{p\}$ and splitting $S \setminus \{p\}$ into two subsets of cardinality $\frac{n}{2}$ and $\frac{n-2}{2}$ for $n$ even and $\frac{n-1}{2}$ each for $n$ odd, respectively. Furthermore, we orient $\ell$ away from $S$: For $H$ a half plane through $p$ containing $S$, the 'head' of $\ell$ lies in the complement of $H$ and the 'tail' of $\ell$ splits $S$.

**Lemma 2** *Let $p$ be an extreme point of $S$ and $\ell$ be a halving ray for $p$. Moving $p$ along $\ell$ in the given orientation, every occurring flip event decreases the number of crossings of $S$.*

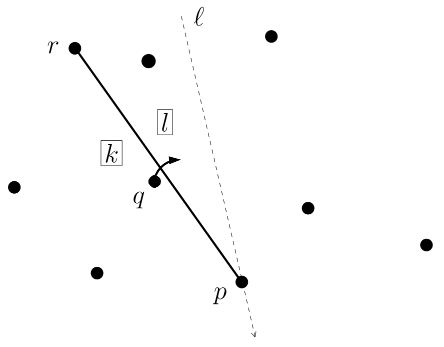


Figure 2: Idea of the proof of Lemma 2.

**Sketch of the Proof.** When point $p$ (see Figure 2) is moved along a halving ray $\ell$, every flip involves $p$ and the center role is played by a different point $q$. Line $\ell$ being a halving ray implies that the flip is a $(k,l)$-flip with $k < l$. $\square$

**Lemma 3** *For every pair of non-consecutive extreme points $p$ and $q$ of $S$, we can choose halving rays that cross in the interior of the convex hull of $S$.*

**Remark 4** Using order type preserving projective transformations it can also be seen that a triangular convex hull can be obtained by projection along the halving ray. This is a rather common tool when working with order types, see e.g. [12]. However, we have decided to use a self-contained, planar approach.

We now have the ingredients to go for our first main results:

**Theorem 5** *Let $S$ be a set of $n$ points in the plane in general position with $h > 3$ extreme points. Then there exists a set $S'$ of $n$ points in general position with fewer crossings than those of $S$ and fewer than $h$ extreme points.*

As a consequence of Theorem 5 we prove the following common belief (see e.g. [7]) for which evidence was provided by all configurations attaining $\overline{cr}(K_n)$ for $n \leq 17$, [2, 4]:

**Theorem 6** *Any set $S$ of $n \geq 3$ points in the plane in general position attaining the rectilinear crossing number has precisely 3 extreme points, that is, a triangular convex hull.*

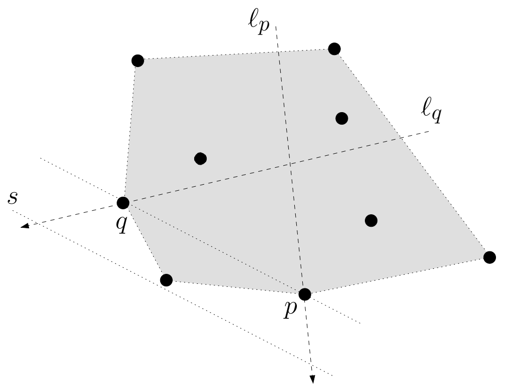


Figure 3: Idea of the proof of Theorem 5.

**Observation 1** If $S$ has 3 extreme points, from our proof of Theorem 5 it follows that for an optimal set $S$ the three extreme points have to be 'far away' in the following sense: For every extreme point $p$ of $S$, the cyclic sorted order of $S \setminus \{p\}$ around $p$ has to be the same as its sorted order in the direction orthogonal to the halving ray of $p$. (Otherwise another flip event would occur when we keep on moving $p$).

## 3 Halving edges, $j$-edges and $k$-sets

A *$k$-set* of $S$ is a set $S' \subset S$ of $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$ points that can be separated from $S \setminus S'$ by a line (hyperplane in general dimension). In dimension 2 there is a one-to-one relation between the numbers of $k$-sets and $(k-1)$-edges, since each of these objects can be derived from precisely two of its corresponding counterparts. Thus, in this paper we will solely use the notion of $j$-edges, although all the results can also be stated in terms of $k$-sets.

A *halving edge* is a $j$-edge with $j = \lfloor \frac{n-2}{2} \rfloor$. Similarly to Lemma 1, we now consider how the numbers of $j$-edges and halving edges change during a flip.

**Lemma 7** *A $(k, l)$-flip changes the number of $j$-edges in the following way: For $k < l$ it decreases the number of $k$-edges by one and increases the number of $(k + 1)$-edges by one. For $k > l$ it decreases the number of $(l+1)$-edges by one and increases the number of $l$-edges by one. It thus changes the number of halving edges by 0 or 1 for $k < l$, 0 or $-1$ for $k > l$. For $k = l$ everything remains unchanged.*

**Lemma 8** *A $(k, l)$-flip either leaves the number of halving edges unchanged or increases it by 1 for $k < l$ (decreases it by 1 for $k > l$, respectively).*

Results similar to Lemmas 7 and 8 have been obtained for dimension 3 in [5]. We are now ready to show our main result for halving edges.

**Theorem 9** *For any fixed $n \geq 3$, there exists a point set with triangular convex hull that maximizes the number of halving edges.*

One might wonder whether we can obtain a stronger result similar to Theorem 6 stating that any point set maximizing the number of halving edges has to have a triangular convex hull. But there exist sets of 8 points with 4 extreme points bearing the maximum of 9 halving edges, see [5], and similar examples exist for larger $n$. Hence, the stated relation is tight in this sense. We leave as an open problem the existence of a constant $h$ such that any point set maximizing the number of halving edges has at most $h$ extreme points. We conjecture that such a constant exists, and the results for $n \leq 11$ suggest that $h = 4$ could be the tight bound.

From Lemma 1 and Lemma 7 we get a relation between the number $\overline{cr}(S)$ of rectilinear crossings of $S$ and the number of $j$-edges of $S$, denoted by $f_j$. An equivalent relation can be found in [13].

**Lemma 10**

$$\overline{cr}(S) + \sum_{j=0}^{\lfloor \frac{n-2}{2} \rfloor} (j-1)(n-j-3) f_j = \frac{n^4 - 10n^3 + 27n^2 - 18n}{8}$$

**Lemma 11** *The number $\overline{cr}(S)$ of rectilinear crossings of $S$ can be computed in $O(n^2)$ time.*

Define the *$j$-edge vector* of $S$ as $(f_0, \ldots, f_{\lfloor \frac{n-2}{2} \rfloor})$. Another consequence of Lemma 10 is that any two sets of $n$ points with the same $j$-edge vector necessarily have the same number of crossings. The reverse is in general not true, as we have examples of sets with 11 points and 106 crossings each, but $j$-edge vectors $(3, 6, 9, 15, 22)$ and $(3, 6, 10, 12, 24)$, respectively. But it is conjectured that for point sets attaining the rectilinear crossing number this relation is in fact a bijection [11]. This conjecture is known to be true for $n \leq 16$, and it turned out that the distribution of the number of $j$-edges follows some interesting patterns, see [4] for details.

Let us consider the flipping operation as some kind of local improvement operation, in order to obtain a global optimum that minimizes the number of crossings. The idea would be to start with an arbitrary point set and to repeat an improving flip until no more improving flips exist. However, as one would expect, there exists an example of 9 points with 40 crossings and a $j$-edge vector $(3, 6, 11, 16)$ which is locally optimal w.r.t. flips. But the global optimum has only 36 crossings and a $j$-edge vector $(3, 6, 9, 18)$.

## 4 On $j$-edge vectors and $(\leq j)$-edges

Recall that a $(\leq j)$-edge is a segment spanned by two points $a, b \in S$ that has at most $j$ points of $S$ in one open half space defined by the line through $ab$.

Using similar notations as above, $f_{(\leq j)}$ counts the number of $(\leq j)$-edges of $S$ and $(f_{(\leq 0)}, \ldots, f_{(\leq \lfloor \frac{n-2}{2} \rfloor)})$ is the $(\leq j)$-*edge vector* of $S$.

**Lemma 12** *Let $S$ be a set of $n$ points with $h > 3$ extreme points, having $(\leq j)$-edge vector $(f_{(\leq 0)}, \ldots, f_{(\leq \lfloor \frac{n-2}{2} \rfloor)})$. Then there exists a set $S'$ of $n$ points with triangular convex hull and $(\leq j)$-edge vector $(f'_{(\leq 0)}, \ldots, f'_{(\leq \lfloor \frac{n-2}{2} \rfloor)})$ with $f'_{(\leq i)} \leq f_{(\leq i)}$ for all $i = 0, \ldots, \lfloor \frac{n-2}{2} \rfloor$, where at least one inequality is strict.*

This allows us to give a geometric proof of the following result, which was proved in [13] using circular sequences:

**Theorem 13** *Let $S$ be a set of $n$ points in the plane. The number of $(\leq j)$-edges of $S$ is at least $3\binom{j+2}{2}$ for $0 \leq j < \frac{n-2}{2}$. This bound is tight for $j \leq \lfloor \frac{n}{3} \rfloor - 1$.*

That Theorem 13 is not tight for $j \geq \lfloor \frac{n}{3} \rfloor$ can be seen for $n = 5$: For any order type, all $\binom{5}{2} = 10$ segments are $(\leq 1)$-edges. Still we can prove some property of the lexicographic minimal $j$-edge vector:

**Corollary 14** *Let $v = (f_0, \ldots, f_{\lfloor \frac{n-2}{2} \rfloor})$ be the lexicographic minimal $j$-edge vector for a set $S$ of $n$ points in the plane. Then, $f_i = 3$ for $i = 0, \ldots, \lfloor \frac{n}{3} \rfloor - 1$ and $f_{\lfloor \frac{n}{3} \rfloor} \geq 3$.*

The investigation of lexicographic minimal $j$-edge vectors is driven by the following conjecture, for which evidence is also provided by Lemma 10:

**Conjecture 1** *Point sets attaining the rectilinear crossing number have a lexicographic minimal $j$-edge vector.*

The next result provides insight about the structure of point sets minimizing the $j$-edge vector. For a set $S$ of $n$ points in the plane we call the convex hull of $S$ its *1-st convex layer*. The *$j$-th convex layer* of $S$ is the convex hull of $S_j$, where $S_j$ is the set of points we get after removing all points from $S$ which lie on the $k$-th convex layer for $1 \leq k < j$.

**Theorem 15** *If $S$ has $3\binom{j+2}{2}$ $(\leq j)$-edges for every $j$, $0 \leq j \leq 2J < \frac{n-2}{2}$, then for $1 \leq k \leq J+1$ the $k$-th convex layer of $S$ is a triangle, consisting of three $(2(k-1))$-edges.*

Finally, from Theorem 13 we know that sets minimizing the $j$-edge vector (or equivalently the $(\leq j)$-edge vector) have precisely $3\binom{j+2}{2}$ $(\leq j)$-edges for $j \leq \lfloor \frac{n}{3} \rfloor - 1$. We thus get:

**Corollary 16** *Let $S$ be a set lexicographically minimizing the $j$-edge vector. Then the outermost $\lfloor \frac{n+3}{6} \rfloor$ convex layers of $S$ are triangles. Among them, the $k$-th convex layer consists of three $(2(k-1))$-edges.*

## References

[1] B.M. Ábrego and S. Fernández-Merchant, *A lower bound for the rectilinear crossing number.* Graphs and Combinatorics, 21:3 (2005), 293–300.

[2] O. Aichholzer, *Rectilinear Crossing Number Page.* `http://www.ist.tugraz.at/staff/aichholzer/crossings.html`

[3] O. Aichholzer, F. Aurenhammer, H. Krasser, *On the crossing number of complete graphs.* Computing 76 (2006) 165–176.

[4] O. Aichholzer, H. Krasser, Abstract Order Type Extension and New Results on the Rectilinear Crossing Number. In *Proc. Proc. 21th Ann. Sympos. Comput. Geom.*, Pisa, Italy (2005), 91–98.

[5] A. Andrzejak, B. Aronov, S. Har-Peled, R. Seidel, E. Welzl, Results on $k$-Sets and $j$-Facets via Continuous Motion. In *Proc. Proc. 14th Ann. Sympos. Comput. Geom.*, Minneapolis, USA, (1998), 192–199.

[6] J. Balogh, G. Salazar, Improved bounds for the number of $(\leq k)$-sets, convex quadrilaterals, and the rectilinear crossing number of $K_n$. In *Proc.12th Int. Symp. on Graph Drawing.* LNCS 3383 (2005), 25–35.

[7] A. Brodsky, S. Durocher, E. Gethner, *Toward the rectilinear crossing number of $K_n$: new drawings, upper bounds, and asymptotics.* Discrete Mathematics 262 (2003), 59–77.

[8] P. Erdös, R.K. Guy, *Crossing number problems.* American Mathematical Monthly 80 (1973), 52–58.

[9] J.E.Goodman, R.Pollack, *Multidimensional sorting.* SIAM J. Computing 12, 484-507, 1983.

[10] R.K. Guy, *A combinatorial problem.* Nabla (Bulletin of the Malayan Mathematical Society) 7 (1960), 68–72.

[11] H.F.Jensen, *Personal communication.* (2004/05)

[12] H.Krasser, *Order Types of Point Sets in the Plane.* PhD-Thesis, TU-Graz (2003)

[13] L.Lovász, K.Vesztergombi, U.Wagner, E.Welzl, *Convex Quadrilaterals and $k$-Sets.* Contemporary Mathematics 342 (2004), 139–148.

[14] J. Pach, G. Tóth, *Thirteen problems on crossing numbers* Geombinatorics 9 (2000), 195–207.

[15] W.T. Tutte, *Toward a theory of crossing numbers.* Journal of Combinatorial Theory 8, (1970), 45–53.

[16] I. Vrt'o, Crossing numbers of graphs: A bibliography. `http://www.ifi.savba.sk/∼imrich`

# On the All-Farthest-Segments Problem for a Planar Set of Points

Asish Mukhopadhyay [*]
e-mail: asishm@cs.uwindsor.ca

Samidh Chatterjee [†]
e-mail: chatte7@cs.uwindsor.ca

Benjamin Lafreniere [‡]
e-mail: lafreni@uwindsor.ca

## Abstract

In this note, we outline a very simple algorithm for the following problem: Given a set $S$ of $n$ points $p_1, p_2, p_3, \ldots, p_n$ in the plane, we have $O(n^2)$ segments implicitly defined on pairs of these $n$ points. For each point $p_i$, find a segment from this set of implicitly defined segments that is farthest from $p_i$. The complexity of our algorithm is in $O(nh + n \log n)$, where $n$ is the number of input points, and $h$ is the number of vertices on the convex hull of $S$.

## 1 Introduction

Geometric optimization is a very active subarea of Computational Geometry. In this paper we study a simple geometric optimization problem. Given a set $S$ of $n$ points $p_1, p_2, p_3, \ldots, p_n$ in the plane, we have $O(n^2)$ segments implicitly defined on pairs of these $n$ points. For each point $p_i$, find a segment from this set of implicitly defined segments that is farthest from $p_i$.

## 2 Previous work

For the *nearest* version of this problem Daescu and Luo [1], presented an $O(n \log n)$ algorithm; Duffy et al [2] presented an $O(n^2)$ algorithm for the *all-nearest* version, and also provided evidence that this might be an $O(n^2)$-hard problem. Daescu and Luo [1] also presented an $O(n \log n)$ for the *farthest* version of this problem. Here we show that the *all-farthest* version of the problem can be solved in $O(nh + n \log n))$ time, where $h$ is the number of vertices on the convex hull of the $n$ points.

While it is hard to provide any practical motivation for problems of this type that does not appear contrived, it is intriguing to know whether the *all-farthest* problem can be solved as efficiently as or faster than the *all-nearest* version.

[*]School of Computer Science, University of Windsor, Windsor,Ontario, Canada
[†]School of Computer Science, University of Windsor
[‡]School of Computer Science, University of Windsor

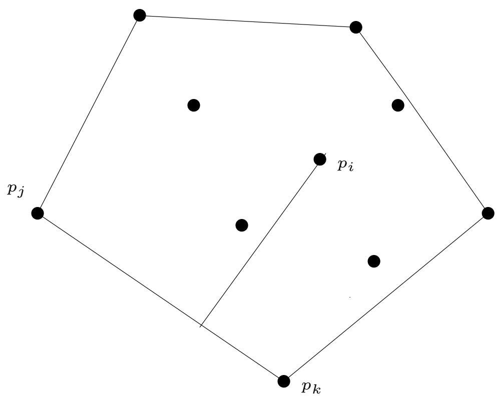


Figure 1: *Farthest distance from* $p_i$ *to segment* $(p_j, p_k)$ *is to an intermediate point*

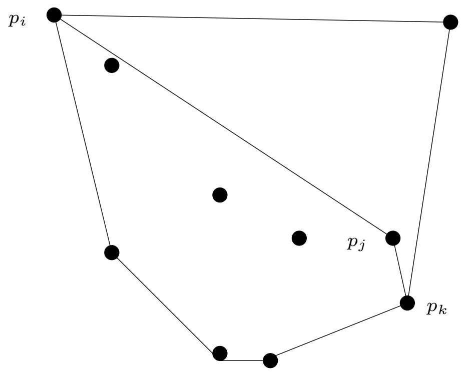


Figure 2: *Farthest distance from* $p_i$ *to segment* $(p_j, p_k)$ *is to an endpoint*

## 3 Characterization of a farthest segment

Let $\overline{p_j p_k}$ be a farthest segment of a point $p_i$. The farthest distance is obtained either by dropping a perpendicular from $p_i$ to the segment $\overline{p_j p_k}$ (Fig. 1) or by joining $p_i$ to the nearer one of the end points $p_j$ and $p_k$ (Fig. 2). We call these two types of farthest segments type $A$ and type $B$ respectively.

We design an algorithm by characterizing the two types of segments. To ensure the correctness of the arguments below, we shall assume that no three points of $S$ are collinear.

**Lemma 1** *If the segment* $\overline{p_j p_k}$ *is a type* $A$ *farthest segment for a point* $p_i$ *then* $\overline{p_j p_k}$ *is an edge on the convex hull of* $S$.

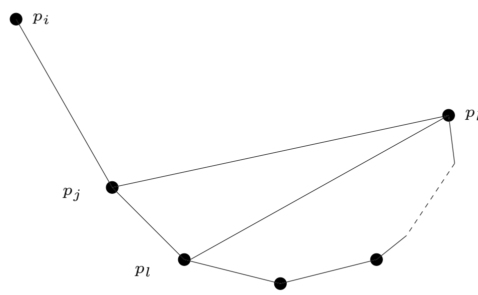


Figure 3: *$p_j$ and $p_k$ are non-adjacent convex hull vertices*

**Proof:** If the segment $\overline{p_jp_k}$ is not a convex hull edge, then there exists a point $p_l$ of $S$ in the open half-plane defined by the supporting line through $p_j$ and $p_k$ that does not contain $p_i$. This gives a segment $\overline{p_jp_l}$ that is farther from $p_i$ than $\overline{p_jp_k}$ since $\overline{p_ip_j}$ is the hypotenuse of the right-triangle formed by $p_i$, $p_j$ and the foot of the perpendicular from $p_i$ to $\overline{p_jp_k}$. This contradicts the assumption that $\overline{p_jp_k}$ is a farthest segment of $p_i$. $\square$

**Lemma 2** *If the segment $\overline{p_jp_k}$ is a type B farthest segment for a point $p_i$ then either $\overline{p_jp_k}$ is an edge on the convex hull of S or $p_j$ is farthest from $p_i$ among all the points that are interior to the convex hull of the point set, while $p_k$ is a convex hull vertex of the given point set (Fig. 2).*

**Proof:** Let the farthest distance be realised by joining $p_i$ to $p_j$. Our proof is in three parts, covering the mutually exclusive and exhaustive possibilities that the end points of $\overline{p_jp_k}$ are both points internal to the convex hull of $S$, are both convex hull vertices or one is an internal vertex while the other is a convex hull vertex.

(1) Suppose $p_j$ and $p_k$ are both internal to the convex hull. If this were true, consider the half-plane defined by a line through $p_k$ orthogonal to $\overline{p_ip_k}$ that does not contain $p_i$. This half plane must contain a vertex $p_l$ of the convex hull of $S$, giving us a segment $\overline{p_kp_l}$ that is farther from $p_i$ than $\overline{p_jp_k}$ and a contradiction. Hence this possibility is excluded.

(2) Suppose $p_j$ and $p_k$ are both vertices of the convex hull. We claim that in this case $\overline{p_jp_k}$ is a convex hull edge. If otherwise, the segment $\overline{p_jp_k}$ divides the convex hull of $S$ into two parts. Consider the convex hull boundary going from $p_j$ to $p_k$ that lies in the part not containing $p_i$ (see Fig. 3). Since there is at least one convex hull vertex on this boundary, let $p_l$ be the one closest to $p_j$. Then $\overline{p_lp_k}$ gives us a segment (could be of type $A$ or Type $B$) that is farther from $p_i$ than $\overline{p_jp_k}$ as the distances of all points on $\overline{p_lp_k}$ from $p_i$ are greater than the distance from $p_i$ to $p_j$. This proves our claim.

(3) $p_k$ is a convex hull vertex and $p_j$ is an internal vertex. We claim that in this case $p_j$ is farthest from $p_i$ among all internal vertices. Otherwise, let $p_l$ be an internal point that is farther from $p_i$ than $p_j$. There exists a point $p_m$ that lies on the convex hull and is in the half-plane defined by a line through $p_l$ orthogonal to $\overline{p_ip_l}$, not containing $p_i$. This gives us a segment $\overline{p_lp_m}$ farther from $p_i$ than $\overline{p_jp_k}$ and a contradiction.

By (1), (2) and (3), we have proven that the farthest segment from $p_i$ must either be an edge of the convex hull of $S$, or have one end point on the convex hull while the other end point is the farthest from $p_i$ among all internal points. $\square$

With these two lemmas, it is easy to design an efficient algorithm for solving this problem.

## 4 Algorithm

We first construct the convex hull of the point set; then the farthest-point Voronoi diagram of the interior points, if any. The complexity of these two steps is in $O(n \log n)$.

For each point $p_i$, we find the farthest segment as outlined in the following algorithm.

```
**Algorithm** All-farthest-segments
*Input*: A set of $n$ points $p_1, p_2, \ldots, p_n$
*Output*: The farthest segment $\overline{p_jp_k}$ for each $p_i$

for each $p_i$ do

Step 1: Find the fathest segment among the edges of the precomputed convex hull; record the segment and the distance.

Step 2: Locate $p_i$ in the precomputed farthest point Voronoi diagram of the points interior to the convex hull. Let $p_j$ be its farthest neighbor and record the distance to it from $p_i$. If this distance is smaller than that computed in Step 1 report the segment found in Step 1 and quit, else continue.

Step 3: Draw a line orthogonal to the segment $\overline{p_ip_j}$; the other endpoint $p_k$ is the convex hull vertex that lies in the halfplane not containing $p_i$. We find this by a linear search (we can afford this!) on the convex hull boundary. Report $\overline{p_jp_k}$ as the farthest segment.

od
```

## 5 Analysis

The complexity of Step 1 is in $O(h)$; that of Step 2 is in $O(\log(n-h))$; while that of Step 3 is also in $O(h)$. Thus the complexity of the all farthest segment is in $O(nh + n\log n)$.

## 6 Future Work

It would be intereresting to extend this algorithm to finding all $k$-th closest segments.

## 7 Acknowledgement

We would like to thank the referees for their perceptive comments. The improvement and clarity of this revised version are due to their comments.

## References

[1] O. Daescu and J. Luo. Proximity problems on line segments spanned by points. In *Proc. of 14th Annual Fall Workshop on Computational Geometry*, pages 9–10, 2004.

[2] H. M. K. Duffy, C. McAloney and D. Rappaport. Closest segments. In *Proc. of CCCG 2005*, pages 229–231, 2005.

# Planar Point Sets with Large Minimum Convex Partitions*

Jesús García-López[†] Carlos M. Nicolás[‡]

## Abstract

Given a finite set $S$ of points in the plane, a *convex partition* of S is a subdivision of the convex hull of $S$ into nonoverlapping empty convex polygons with vertices in $S$. Let $G(S)$ be the minimum $m$ such that there exists a convex partition of $S$ with at most $m$ faces. Let $F(n)$ be the maximum value of $G(S)$ among all the sets of $n$ points in the plane. It is known [1] that $F(n) \geq n+2$ for $n \geq 13$. In this paper we show that, for $n \geq 4$

$$F(n) > \frac{12}{11}n - 2$$

Also, for $n \geq h \geq 3$, let $F_h(n)$ be the maximum value of $G(S)$ among all the sets of $n$ points in which exactly $h$ of them lie in the boundary of the convex hull. We show that

$$F_h(n) > n - \frac{15}{8}h + \sqrt{\frac{(n-h)h}{2}}$$

## 1 Introduction

Triangulations of point sets are one of the most studied structures in computational geometry. More general structures are easily obtained by relaxing some of the conditions that define triangulations. Let a *convex partition* of a set of points $S$ in the plane be a decomposition of the convex hull of $S$ into nonoverlapping convex polygons with any number of edges, such that no point of $S$ belongs to the interior of any of the convex polygons. Equivalently, a convex partition is a tesselation of the points into empty convex polygons covering the entire convex hull of $S$. Intuitively, $S$ should admit convex partitions with considerably fewer faces (polygons) than those in a triangulation, and this may be useful in applications where the complexity depends on the number of faces.

Therefore, we should try to find the minimum number $F(n)$ such that every set of $n$ points in the plane has a convex partition with at most $F(n)$ faces.

*Partially supported by grant MCYT TIC2002-01541

[†]Departamento de Matemática Aplicada. Escuela U. de Informática. Universidad Politécnica de Madrid, Madrid, Spain. email:jglopez@eui.upm.es

[‡]Department of Mathematics, University of Kentucky, Lexington, KY, USA. email:cnicolas@ms.uky.edu

In the open-problem session of CCCG-1998, J. Urrutia [5] conjectured that $F(n) \leq n+1$. Later, in 2001, O. Aichholzer and H. Krasser [1] showed that $F(n) \geq n+2$ (for $n \geq 13$). The best known upper bound, $F(n) \leq \frac{10n-18}{7}$, is due to V. Neumann-Lara et al. [4]. Other related work on convex partitions include the study of simultaneous flips (see [3]) and an algorithm by T. Fevens et al. [2] that computes in polynomial time the minimum convex partition provided that the points lie on the boundaries of a fixed number of nested convex hulls.

## 2 Definitions and Strict Monotonicity

For a finite set $S$ of points in the plane, let $G(S)$ be the number of faces in a convex partition of $S$ with a minimum number of faces.

For $n \geq h \geq 3$, let $F_h(n)$ be the maximum value of $G(S)$ among all the sets $S$ with $n$ points of which exactly $h$ are *extreme*, i.e., lie on the boundary of the convex hull.

Let $F(n)$, $n \geq 3$, be the maximum value of $G(S)$ when $|S| = n$, so $F(n) = \max\{F_h(n) : 3 \leq h \leq n\}$.

The following results show that the functions $F_h(n)$ and $F(n)$ are strictly increasing.

**Proposition 1** *$F_h(n+k) \geq F_h(n)+k$, for $3 \leq h \leq n$, $0 \leq k$.*

**Proof.** It is enough to show that $F_h(n+1) > F_h(n)$.

Let $S$ be a set of $n$ points, $h$ of them extreme, such that $G(S) = F_h(n)$. Let $p$, $p'$ be contiguous extreme points of $S$, see Figure 1. Take $q \notin S$ satisfying

(i) $q$ is in the interior of the convex hull of $S$.

(ii) $q$ is an extreme point of $S-\{p\}$ and $S-\{p'\}$.

(iii) for every $r_1, r_2 \in S-\{p\}$, $p$ and $q$ lie on the same side of the line through $r_1$ and $r_2$.

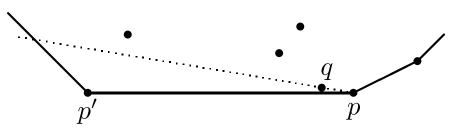


Figure 1: Points as in Proposition 1

Conditions (i) and (ii) imply that every convex partition of $S \cup \{q\}$ contains the triangle with vertices $p, q, p'$ as one of its faces. And (iii) implies that every convex partition of $S \cup \{q\}$ can be transformed into a convex partition of $S$ by first joining to $p$ every vertex adjacent to $q$ and then removing $q$. Since the triangle $pqp'$ is transformed into the edge $pp'$, the result follows by performing this transformation to a minimum convex partition of $S \cup \{q\}$. □

**Corollary 2** *$F(n+k) \geq F(n) + k$, for $n \geq 3, k \geq 0$.*

## 3 The Arrangement

Given two points $o_1, o_2$ located on the $y$-axis, it is not difficult to show by induction on $k \geq 0$ that it is possible to arrange $k(k+1)/2$ points $\{p_{i,j}\}_{1 \leq i \leq j \leq k}$ on $k$ vertical layers $V_j = \{p_{1,j}, p_{2,j}, \ldots, p_{j,j}\}$ and $k$ horizontal layers $H_i = \{p_{i,i}, p_{i,i+1}, \ldots, p_{i,k}\}$, as schematically shown in Figure 2, in such a way that the following properties hold:

(1) Each horizontal layer $H_i$ is concave upward and the $y$-coordinate of $p_{i,j}$ decreases as $j$ increases.

(2) Each vertical layer $V_i$ is concave to the left and the $x$-coordinate of $p_{i,j}$ increases with $i$.

(3) Every point in $V_{j+2}$ lies above the line through $p_{i,j}$ and $p_{i,j+1}$, for $i \in \{1, \ldots, j\}$, $j < k-1$.

(4) Every point in $V_{j+1}$ lies below the line through $p_{i,i}$ and $p_{i+1,j}$, for $i \in \{1, \ldots, k-2\}$, $i < j$.

(5) $V_{j+1}$ lies below the line through $o_1$ and $p_{1,j}$ and above the line through $o_2$ and $p_{j,j}$, for $j < k$.

Additionally, let $A$ be any nonempty set of points satisfying:

(3') $A$ lies above the line through $p_{i,k-1}$ and $p_{i,k}$, $i < k$.

(4') $A$ lies below the line through $p_{i,i}$ and $p_{i+1,k}$, $i < k$.

(5') $A$ lies below the line through $o_1$ and $p_{1,k}$ and above the line through $o_2$ and $p_{k,k}$.

Let $S = \{o_1, o_2\} \cup \{p_{i,j}\}_{1 \leq i \leq j \leq k} \cup A$, and take any convex partition $\Pi$ of $S$.

For $1 \leq i \leq j \leq k$, let $C_{i,j}$ be the convex polygon in $\Pi$ located immediately below $p_{i,j}$, so $C_{i,j}$ is the unique polygon in the partition containing the point $p_{i,j} - (0, \varepsilon)$, for very small $\varepsilon$. Note that $C_{i,k}$ is well defined because $A$ is not empty.

**Proposition 3** *$C_{i_1,j_1} \neq C_{i_2,j_2}$ if $(i_1, j_1) \neq (i_2, j_2)$.*

**Proof.** By contradiction.

Suppose $C_{i_1,j_1} = C_{i_2,j_2} = C$ with $(i_1, j_1) \neq (i_2, j_2)$. Clearly, both $p_{i_1,j_1}$ and $p_{i_2,j_2}$ are upper extreme points of $C$, they lie on the upper boundary of the convex hull of $C$. Without loss of generality assume that $p_{i_1,j_1}$ appears before $p_{i_2,j_2}$ when the extreme upper points of $C$ are listed from left to right. This assumption implies that $j_1 \leq j_2$.

Note that $p_{i_1,j_1}$ cannot be the leftmost point among the extreme upper points of $C$, because then the other points of $C$ belong to $V_{j_1+1} \cup \ldots \cup V_k \cup A$ (recall the x-coordinates of points in $V_{j_1}$ below $p_{i_1,j_1}$ are less than the x-coordinate of $p_{i_1,j_1}$), but then $C$ cannot be the polygon immediately below $p_{i_1,j_1}$.

Let $l$ be the leftmost upper extreme point of $C$, so we know $l \neq p_{i_1,j_1}$ and $l \in \{o_1\} \cup V_1 \cup \ldots \cup V_{j_1}$. Similarly, the rightmost point $r$ among the extreme upper points of $C$ cannot be $p_{i_2,j_2}$, so it belongs to $V_{j_2+1} \cup \ldots \cup V_k \cup A$.

By (2) we see that necessarily $j_1 < j_2$, since if $j_1 = j_2$ then $l$ cannot lie below the line through $p_{i_1,j_1}$ and $p_{i_2,j_2}$. Also, it is clear that $i_1 < i_2$ since if $i_1 \geq i_2$ then $r$ cannot lie below the line through $p_{i_1,j_1}$ and $p_{i_2,j_2}$, by (3).

If $l = o_1$ then (5) and (5') imply that $p_{1,j_2}$ lies above the line through $o_1$ and $r$, but $i_2 > i_1 \geq 1$, so $p_{1,j_2}$ lies in the interior of $C$, a contradiction.

If $l = p_{i_0,j_0}$, $j_0 \leq j_1$, then again by (3) we must have $i_0 < i_1$. Therefore, by (4), $p_{i_2-1,j_2}$ lies above the line through $p_{i_0,j_0}$ and $r$, since $i_0 < i_2 - 1$ (because $i_0 < i_1 < i_2$), so $p_{i_2-1,j_2}$ belongs to the interior of $C$, a contradiction. □

A similar argument shows that the sets $C_{i,j}$ cannot contain any point in $A$, unless $j = k$.

**Proposition 4** *$C_{i,j} \cap A = \emptyset$ if $j < k$.*

Note that from (4') it follows that $o_2$ is the only point above the line through $p_{i,i}$ and $p_{i+1,i+1}$, there-

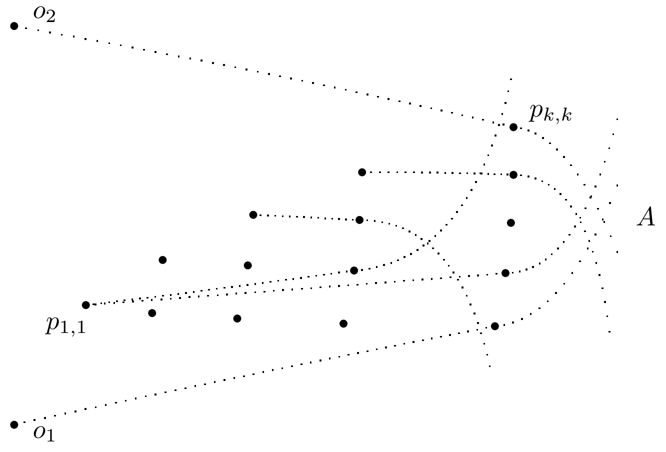


Figure 2: The Arrangement of Points. $X$-coordinates are not to scale, dotted lines represent straight lines.

fore, the edge joining $p_{i,i}$ and $o_2$ belongs to $\Pi$. Let $D_i$, $1 \le i \le k$, be the convex polygon in $\Pi$ to the left of this edge when traversed from $p_{i,i}$ to $o_2$. Using (5) and (5') it is easy to prove the following proposition.

**Proposition 5** *$D_l \neq C_{i,j}$ and $D_l \cap A = \emptyset$, for $1 \le l \le k$, $1 \le i \le j \le k$.*

Propositions 3 and 5 imply that $F_3(n) - n$ goes to infinity as $n$ increases. For future reference, we compute the bound on $F_3$ obtained when $|A| = 1$ and $k = 3, 5, 6$.

**Observation 1** $F_3(9) \ge 9$, $F_3(18) \ge 20$, $F_3(24) \ge 27$.

## 4 Generalization

Now we want to show that, for any $h \ge 3$, it is possible to put together $h$ sets of $k(k+1)/2$ points in such a way that for each $t \in \{1, \ldots, h\}$, the set $\{p^t_{i,j}\}_{1 \le i \le j \le k}$ satisfies properties (1)-(5) and $\cup_{s \neq t}\{p^s_{i,j}\}$, playing the role of $A$, satisfies (3')-(5').

Let $h \ge 3$. Take a regular $h$-gon with vertices $q_1, \ldots, q_h$ (in clockwise order). Assume that $q_1$ and $q_2$ lie on the $y$-axis. Now construct the set $\{o_1, o_2\} \cup \{p_{i,j}\}_{1 \le i \le j \le k}$ from the previous setion taking $\{o_1, o_2\}$ to be $\{q_1, q_2\}$. In addition, require that the projection of every $p_{i,j}$ on the $y$-axis falls in the middle section of the segment $q_1q_2$ when divided into three equal parts. Now compress the $x$-coordinates of the points $p_{i,j}$, i.e., multiply them by a constant, until they are so close to the segment $q_1q_2$ that all the lines mentioned in properties (1)-(5) are so slanted that they intersect the contiguous sides $q_2q_3$ and $q_hq_1$ in the third section adjacent to $q_1q_2$. Call these points $\{p^1_{i,j}\}$ and do the same in each side $q_tq_{t+1}$ (where $q_{h+1} = q_1$) to obtain the points $\{p^t_{i,j}\}$.

Let $S_{h,k} = \{q_i\}_{1 \le i \le h} \cup \{p^t_{i,j}\}^{1 \le t \le h}_{1 \le i \le j \le k}$.

Given any convex partition of $S_{h,k}$, define $C^t_{i,j}$ and $D^t_i$ as in the previous section, for each set $\{q_t, q_{t+1}\} \cup \{p^t_{i,j}\}_{1 \le i \le j \le k}$.

The construction of $S_{h,k}$ allows the set $\cup_{s \neq t}\{p^s_{i,j}\} \cup \{q_i\}_{i \neq t, t+1}$ to play the role of $A$ in Propositions 4 and 5. Therefore, Propositions 3, 4 and 5 imply the following result.

**Proposition 6** *For all $1 \le t_1, t_2 \le h$*

*(i) $C^{t_1}_{i_1,j_1} \neq C^{t_2}_{i_2,j_2}$ if $1 \le i_1 \le j_1 < k$, $1 \le i_2 \le j_2 < k$ and $(i_1, j_1, t_1) \neq (i_2, j_2, t_2)$.*

*(ii) $C^{t_1}_{i,j} \neq D^{t_2}_l$ if $1 \le i \le j \le k$ and $1 \le l \le k$.*

*(iii) $D^{t_1}_{l_1} \neq D^{t_2}_{l_2}$ if $1 \le l_1, l_2 \le k$ and $(l_1, t_1) \neq (l_2, t_2)$.*

## 5 Analysis

Let $n_{h,k}$ be the number of points in $S_{h,k}$, so

$$n_{h,k} = h + h\frac{k(k+1)}{2} \tag{1}$$

Take any convex partition of $S_{h,k}$. By the last proposition, the polygons $C^t_{i,j}$, $D^t_i$ are all distinct if $j < k$. This adds up to $hk(k-1)/2 + hk = hk(k+1)/2$ faces. Now, each point $p^t_{i,k}$ must be joined to some point not in $\{q_t, q_{t+1}\} \cup \{p^t_{i,j}\}$, because by property (2) there is a line through $p^t_{i,k}$ that contains no point of this set. Pick an edge going from $r = p^t_{i,k}$ to a point $s$ in $(\{q_{t'}, q_{t'+1}\} \setminus \{q_t, q_{t+1}\}) \cup \{p^{t'}_{i,j}\}$, $t' \neq t$, and let $E_{rs}$ be the convex set to the left of this edge oriented from $r$ to $s$ if $t < t'$ and form $s$ to $r$ in the other case. Clearly, each $E_{rs}$ is different from every $C$ and $D$ polygon that we counted above and from every $E_{r's'}$ polygon associated with other edge $r's' \neq rs$. Moreover, each $rs$ edge appears at most twice, so we have at least $hk/2$ additional polygons in the convex partition. Hence,

$$G(S_{h,k}) \ge \frac{hk(k+1)}{2} + \frac{hk}{2} = n_{h,k} - h + \frac{hk}{2} \tag{2}$$

Now we think of $h$ as fixed and write $n_k$ in place of $n_{h,k}$. Solving for $k$ in (1) and substituting in (2) gives

$$\begin{aligned} G(S_{h,k}) &= n_k - h + \frac{h}{2}\left(\frac{-1 + \sqrt{1 + \frac{8(n_k - h)}{h}}}{2}\right) \\ &> n_k - \frac{5}{4}h + \sqrt{\frac{(n_k - h)h}{2}} \end{aligned}$$

Hence, $F_h(n_k) > n_k - \frac{5}{4}h + \sqrt{\frac{(n_k-h)h}{2}}$. Now we derive a formula valid for every $n$.

**Proposition 7** *$F_h(n) > n - \frac{15}{8}h + \sqrt{\frac{(n-h)h}{2}}$, for $n \ge h$.*

**Proof.** Let $k$ be an integer such that $n_k \le n < n_{k+1}$. By Proposition 1, $F_h(n) \ge F_h(n_k) + n - n_k > n - \frac{5}{4}h + \sqrt{\frac{(n_k-h)h}{2}}$. Now, $\sqrt{(n-h)h/2} - \sqrt{(n_k-h)h/2} < \sqrt{(n_{k+1}-h)h/2} - \sqrt{(n_k-h)h/2} = h\sqrt{k+1}/(\sqrt{k+2} + \sqrt{k})$. But $\sqrt{x+1}/(\sqrt{x+2} + \sqrt{x})$ decreases quickly to $1/2$ and for $x = 1$ its value is less than $5/8$, so the result follows for $k \ge 1$. The case $k = 0$ can be verified separately, using that in this situation $h \le n < 2h$ and $F_h(n) \ge (3/2)(n-h) + 1$. □

The best lower bound for $F(n)$ that can be obtained by means of the sets $S_{h,k}$ is attained when $k = 5$ and $h$ varies. Since $n_{h,5} = 16h$, using formula (2) we get $G(S_{h,5}) \ge (35/32)n_{h,5} > (12/11)n_{h,5}$, $h \ge 3$. Therefore, taking into account the monotonicity, we get

**Proposition 8** *$F(n) > \frac{12}{11}n - 2$, for $n \geq 4$.*

**Proof.** If $n \geq 48$, say $16h \leq n < 16(h+1)$, use the fact $F(16h) > (12/11)16h$ and Corollary 2. If $4 \leq n < 48$, the result follows from Corollary 2, the obvious result $F(4) = 3$ and Observation 1. $\square$

## References

[1] O.Aichholzer and H.Krasser. The point set order type data base: A collection of applications and results. *Proc. 13th Annual Canadian Conference on Computational Geometry*, Waterloo, Canada, (2001).

[2] T. Fevens, H. Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, Vol. 109, pp. 95-107, (2001).

[3] H. Meijer and D. Rappaport. Simultaneous Edge Flips for Convex Subdivisions. *Proc. 16th Canadian Conference on Computational Geometry*, Montreal, Canada, (2004).

[4] V. Neumann-Lara, E. Rivera-Campo, and J. Urrutia. A note on convex decompositions of a set of points in the plane. *Graphs and Combinatorics*, Vol. 20, no. 2, pp. 223-231, (2004).

[5] J. Urrutia, Open-problem session, *10th Canadian Conference on Computational Geometry*, Montreal, Canada, (1998).

# Kinetic Collision Detection for Balls Rolling on a Plane

Mohammad Ali Abam* Mark de Berg* Sheung-Hung Poon* Bettina Speckmann*

**Abstract**

This abstract presents a first step towards kinetic collision detection in 3 dimensions. In particular, we design a compact and responsive kinetic data structure (KDS) for detecting collisions between $n$ balls of arbitrary sizes rolling on a plane. The KDS has size $O(n \log n)$ and can handle events in $O(\log n)$ time. The structure processes $O(n^2)$ events in the worst case, assuming that the objects follow low-degree algebraic trajectories. The full paper [1] presents additional results for convex fat 3-dimensional objects that are free-flying in $\mathbb{R}^3$.

## 1 Introduction

Collision detection is a basic computational problem arising in all areas of computer science involving objects in motion—motion planning, animated figure articulation, computer simulated environments, or virtual prototyping, to name a few. Very often the problem of detecting collisions is broken down into two phases: a *broad phase* and a *narrow phase*. The broad phase determines pairs of objects that might possibly collide, frequently using (hierarchies of) bounding volumes to speed up the process. The narrow phase then uses specialized techniques to test each candidate pair, often by tracking closest features of the objects in question, a process that can be sped up significantly by exploiting spatial and temporal coherence. See [13] for a detailed overview of algorithms for such collision and proximity queries.

Algorithms that deal with objects in motion traditionally discretize the time axis and compute or update their structures based on the position of the objects at every time step. But since collisions tend to occur rather irregularly it is nearly impossible to choose the perfect time-step: too large an interval between sampled times will result in missed collisions, too small an interval will result in unnecessary computations (and still there is no guarantee that no collisions are missed). Event-driven methods, on the other hand, compute the event times of significant changes to a system of moving objects, store those in a priority queue sorted by time, and advance the system to the event at the front of the queue. The kinetic data structure (KDS) framework initially introduced by Basch et al. [4] presents a systematic way to design and analyze event-driven data structures for moving objects (see [7] and [8] for surveys on kinetic data structures).

A kinetic data structure is designed to maintain or monitor a discrete attribute of a set of moving objects, where each object has a known motion trajectory or *flight plan*. A KDS contains a set of *certificates* that constitutes a proof of the property of interest. These certificates are inserted in a priority queue (*event queue*) based on their time of expiration. The KDS then performs an event-driven simulation of the motion of the objects, updating the structure whenever a certificate fails. A KDS for collision detection finds a set of geometric tests (elementary certificates) that together provide a proof that the input objects are disjoint.

Kinetic data structures and their accompanying maintenance algorithms can be evaluated and compared with respect to four desired characteristics. A good KDS is *compact* if it uses little space in addition to the input, *responsive* if the data structure invariants can be restored quickly after the failure of a certificate, *local* if it can be updated easily when the flight plan for an object changes, and *efficient* if the worst-case number of events handled by the data structure for a given motion is small compared to some worst-case number of "external events" that must be handled for that motion.

**Kinetic data structures for collision detection.** One of the first papers on kinetic collision detection was published by Basch et al. [3], who designed a KDS for collision detection between two simple polygons in the plane. Their work was extended to an arbitrary number of polygons by Agarwal et al. [2]. Kirkpatrick et al. [11] and Kirkpatrick and Speckmann [12] also described KDS's for kinetic collision detection between multiple polygons in the plane. These solutions all maintain a decomposition of the free space between the polygons into "easy" pieces (usually pseudo-triangles). Unfortunately it seems quite hard to define a suitable decomposition of the free space for objects in 3D, let alone maintain it while the objects move—the main problem being, that

*Department of Mathematics and Computer Science, TU Eindhoven, m.a.abam@tue.nl, m.t.d.berg@tue.nl, spoon@win.tue.nl, speckman@win.tue.nl. M.A. and S.-H.P. were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 612.065.307. M.d.B. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

all standard decomposition schemes in 3D can have quadratic complexity. Hence, even though collision detection is the obvious application for kinetic data structures, there has hardly been any work on kinetic collision detection in 3D.

There are only a few papers that deal directly with (specialized versions of) kinetic 3D collision detection. Guibas et al. [9], extending work by Erickson et al. [6] in the plane, show how to certify the separation of two convex polyhedra moving rigidly in 3D using certain outer hierarchies. Basch et al. [5] describe a structure for collision detection among multiple convex *fat* objects that have almost the same size. The structure of Basch et al. uses $O(n \log^2 n)$ storage and events can be processed in $O(\log^3 n)$ time. If all objects are spheres of related sizes Kim et al. [10] present an event-driven approach that subdivides space into cells and processes events whenever a sphere enters or leaves a cell. Unfortunately there is only experimental evidence for the performance of this structure. Finally, Guibas et al. [9] use the power diagram of a set of arbitrary balls in 3D to kinetically maintain the closest pair among them. The worst-case complexity of this structure is quadratic and it might undergo more than cubically many changes.

**Results.** In this abstract we describe a compact and responsive kinetic data structure for detecting collisions between $n$ balls of arbitrary sizes rolling on a plane. The KDS has size $O(n \log n)$ and can handle events in $O(\log n)$ time. It processes $O(n^2)$ events in the worst case, assuming that the objects follow low-degree algebraic trajectories.

## 2 Balls rolling on a plane

Assume that we are given a set $\mathcal{B}$ of $n$ 3-dimensional balls which are rolling on a 2-dimensional plane $T$, that is, the balls in $\mathcal{B}$ move continuously while remaining tangent to $T$. In this section we describe a responsive and compact KDS that detects collisions between the balls in $\mathcal{B}$.

The basic idea behind our KDS is to construct a *collision tree* recursively as follows:

- If $|\mathcal{B}| = 1$, then there are obviously no collisions and the collision tree is just a single leaf.
- If $|\mathcal{B}| > 1$, then we partition $\mathcal{B}$ into two subsets, $\mathcal{B}_S$ and $\mathcal{B}_L$. The subset $\mathcal{B}_S$ contains the $\lfloor n/2 \rfloor$ smallest balls and the subset $\mathcal{B}_L$ contains the $\lceil n/2 \rceil$ largest balls from $\mathcal{B}$, where ties are broken arbitrarily. The collision tree now consists of a root node that has an associated structure to detect collisions between any ball from $\mathcal{B}_S$ and any ball from $\mathcal{B}_L$, and two subtrees that are collision trees for the sets $\mathcal{B}_S$ and $\mathcal{B}_L$, respectively.

To detect all collisions between the balls in $\mathcal{B}$ it suffices to detect collisions between the two subsets maintained at every node of the collision tree. Let $\mathcal{B}_S$ and $\mathcal{B}_L$ denote the two subsets maintained at a particular node. The remainder of this section focusses on detecting collisions between the balls contained in $\mathcal{B}_S$ and $\mathcal{B}_L$. In particular, we describe a KDS of size $O(|\mathcal{B}_S| + |\mathcal{B}_L|)$ that can handle events in $O(1)$ time—see Theorem 5. The structure processes $O((|\mathcal{B}_S| + |\mathcal{B}_L|)^2)$ events in the worst case, assuming that the balls follow low-degree algebraic trajectories. Since the same event can occur simultaneously at $O(\log n)$ nodes of the collision tree, we obtain the following theorem:

**Theorem 1** *For any set $\mathcal{B}$ of $n$ 3-dimensional balls that roll on a plane, there is a KDS for collision detection that uses $O(n \log n)$ space and processes $O(n^2)$ events in the worst case, assuming that the balls follow low-degree algebraic trajectories. Each event can be handled in $O(\log n)$ time.*

### 2.1 Detecting collisions between small and large balls

As mentioned above, we can restrict ourselves to detecting collisions between balls from two disjoint sets $\mathcal{B}_S$ and $\mathcal{B}_L$ where the balls in $\mathcal{B}_L$ are at least as large as the balls in $\mathcal{B}_S$. Recall that all balls are rolling on a plane $T$. Our basic strategy is the following: we associate a region $D_i$ on $T$ with each $B_i \in \mathcal{B}_L$ such that if the point of tangency of a ball $B_j \in \mathcal{B}_S$ and $T$ is not contained in $D_i$, then $B_j$ can not collide with $B_i$. The regions associated with the balls in $\mathcal{B}_L$ need to have two important properties: $(i)$ each point in $T$ is contained in a constant number of regions and $(ii)$ we can efficiently detect whenever a region starts or stops to contain a tangency point when the balls in $\mathcal{B}_L$ and $\mathcal{B}_S$ move. We first deal with the first requirement, that is, we consider $\mathcal{B}_L$ to be static. For a ball $B_i$ let $r_i$ denote its radius and let $t_i$ be the point of tangency of $B_i$ and $T$.

**The threshold disk.** We define the distance of a point $q$ in the plane to a ball $B_i$ as follows. Imagine that we place a ball $B(q)$ of initial radius 0 at point $q$. We then inflate $B(q)$ while keeping it tangent to $T$ at $q$, until it collides with $B_i$. The radius of $B(q)$ equals the distance of $q$ and $B_i$ which we denote by $\mathrm{dist}(q, B_i)$. More precisely, $\mathrm{dist}(q, B_i)$ is the radius of the unique ball that is tangent to $T$ at $q$ and tangent to $B_i$. It is easy to show that $\mathrm{dist}(q, B_i) = d(q, t_i)^2/4r_i$ where $d(q, t_i)$ denotes the Euclidean distance between $q$ and $t_i$.

Since we have to detect collisions only with balls from $\mathcal{B}_S$ we can stop inflating when $B(q)$ is as large as the smallest ball in $\mathcal{B}_L$. Based on this, we define the threshold disk $D_i$ of a ball $B_i \in \mathcal{B}_L$ as follows: a point $q \in T$ belongs to $D_i$ if and only if $\text{dist}(q, B_i) \leq r_{\min}$ where $r_{\min}$ is the radius of the smallest ball in $\mathcal{B}_L$. It is straightforward to show that $D_i$ is a disk whose radius is $2\sqrt{r_i \cdot r_{\min}}$ and whose center is $t_i$.

Clearly a ball $B_j \in \mathcal{B}_S$ can not collide with a ball $B_i \in \mathcal{B}_L$ as long as $t_j$ is outside $D_i$. In following, we prove that a point $q \in T$ can be contained in at most a constant number of threshold disks. For a given constant $c \geq 0$ let us denote with $c \cdot D_i$ a disk with radius $c \cdot \text{radius}(D_i)$ and center $t_i$.

**Lemma 2** *The number of disks $D_j$ that are at least as large as a given disk $D_i$ and for which $c \cdot D_i \cap c \cdot D_j \neq \emptyset$, is at most $(8\,c^2 + 2\,c + 1)^2 + 1$.*

**Proof.** Let $\mathcal{D}(i)$ be the set of all disks $D_j$ that are at least as large as $D_i$ and for which $c \cdot D_i \cap c \cdot D_j \neq \emptyset$. First we prove that there are no two balls $B_j$ and $B_k$ such that $r_k \geq r_j > 16\,c^2\,r_i$ and $D_j, D_k \in \mathcal{D}(i)$. Assume, for contradiction, that there are two balls $B_j$ and $B_k$ such that $r_k \geq r_j > 16\,c^2\,r_i$ and $D_j, D_k \in \mathcal{D}(i)$. Since $B_j$ and $B_k$ are disjoint, we have

$$d(t_j, t_k) \geq 2\sqrt{r_j \cdot r_k} > 8\,c\sqrt{r_k \cdot r_i}\,.$$

On the other hand, we know that

$$\begin{aligned} d(t_j, t_k) &\leq d(t_j, t_i) + d(t_i, t_k) \\ &\leq (2\,c\sqrt{r_i \cdot r_{\min}} + 2\,c\sqrt{r_j \cdot r_{\min}}) + \\ &\quad (2\,c\sqrt{r_i \cdot r_{\min}} + 2\,c\sqrt{r_k \cdot r_{\min}}) \\ &< 8\,c\sqrt{r_k \cdot r_i} \end{aligned}$$

which is a contradiction. Hence, there is at most one ball $B_j$ such that $r_j > 16\,c^2\,r_i$ and $D_j \in \mathcal{D}(i)$.

It remains to show that the number of balls $B_j$ whose radii are not greater than $16\,c^2\,r_i$ and whose disks $D_j$ belong to $\mathcal{D}(i)$ is at most $(8\,c^2 + 2\,c + 1)^2$. Let $B_j$ be one of these balls and let $x$ be a point in $c \cdot D_j \cap c \cdot D_i$. Since

$$\begin{aligned} d(t_i, t_j) &\leq d(t_i, x) + d(t_j, x) \\ &\leq 2\,c\sqrt{r_i \cdot r_{\min}} + 2\,c\sqrt{r_j \cdot r_{\min}} \\ &\leq (2\,c + 8\,c^2)\,r_i \end{aligned}$$

$t_j$ must lie in a disk whose center is $t_i$ and whose radius is $(2\,c + 8\,c^2)\,r_i$. We also know that for any two such balls $B_j$ and $B_k$, $d(t_j, t_k) \geq 2\sqrt{r_j \cdot r_k} \geq 2\,r_i$ holds. Thus the set $\mathcal{D}'(i)$ of disks centered at $t_j$ with radius $r_i$ for all $D_j \in \mathcal{D}(i)$ are disjoint. Note that any disk in $\mathcal{D}'(i)$ lies inside the disk centered at $t_i$ with radius $((2\,c + 8\,c^2) + 1)\,r_i$. Thus $|\mathcal{D}(i)| = |\mathcal{D}'(i)| \leq \pi\,((2\,c + 8\,c^2 + 1)\,r_i)^2/\pi r_i^2 = (2\,c + 8\,c^2 + 1)^2$. □

**Lemma 3** *Each point $q \in T$ is contained in at most a constant number of threshold disks.*

**Proof.** Let $D_i$ be the smallest threshold disk containing $q$. Lemma 2 with $c = 1$ implies that the number of disks which are not smaller than $D_i$ and which intersect $D_i$ is constant. Hence the number of threshold disks containing $q$ is constant. □

The threshold disks have the important property that each point in $T$ is contained in a constant number of disks. But unfortunately, as the balls in $\mathcal{B}_L$ and $\mathcal{B}_S$ move, it is difficult to detect efficiently whenever a tangency point enters or leaves a threshold disk. Hence we replace each threshold disk by its axis-aligned bounding box. The bounding box of a threshold disk $D_i$ associated with a $B_i \in \mathcal{B}_L$ is called a *threshold box* and is denoted by $\text{TB}(B_i)$. In the following we prove that the threshold boxes retain the crucial property of the threshold disk, namely, that each point $q \in T$ is contained in at most a constant number of threshold boxes.

**Lemma 4** *Each point $q \in T$ is contained in at most a constant number of threshold boxes.*

**Proof.** Instead of considering the threshold boxes directly, we consider the disks defined by the circumcircles $D(\text{TB}(B_j))$ of each threshold box $\text{TB}(B_j)$ with $B_j \in \mathcal{B}_L$. Clearly we have $\text{radius}(D(\text{TB}(B_j))) = \sqrt{2} \cdot \text{radius}(D_j)$ for all $B_j \in \mathcal{B}_L$. Let $\text{TB}(B_i)$ be the smallest box containing $q$. Lemma 2 with $c = \sqrt{2}$ implies that the number of circumcircle disks which are at least as large as $D(\text{TB}(B_i))$ and which intersect $D(\text{TB}(B_i))$ is constant. Hence the number of threshold boxes which are not smaller than $\text{TB}(B_i)$ and intersect $\text{TB}(B_i)$ is constant and so is the number of threshold boxes containing $q$. □

**Kinetic maintenance.** Recall that to detect collisions between $\mathcal{B}_S$ and $\mathcal{B}_L$, for each ball $B_j \in \mathcal{B}_S$ we determine which threshold boxes contain the tangency point $t_j$. Note that according to Lemma 4, $t_j$ is contained in a constant number of threshold boxes. For each $B_j \in \mathcal{B}_S$ we maintain the set of threshold boxes that contain $t_j$ and certificates that guarantees disjointness of $B_j$ and the balls from $\mathcal{B}_L$ whose threshold boxes contain $t_j$.

To maintain our structure we only need to detect when a tangency point $t_j$ enters or leaves a threshold box. To do so, we maintain two sorted lists on the $x$- and $y$-coordinates of the tangency points of $\mathcal{B}_S$ and the extremal points of the threshold boxes associated with the balls in $\mathcal{B}_L$. Clearly the number of events processed by our structure is quadratic in the size of of $\mathcal{B}_S$ and $\mathcal{B}_L$ and each event can be processed in constant time. Unfortunately this structure is not local—a ball $B_i \in \mathcal{B}_L$ might be involved in a number of certificates that is linear in the size of $\mathcal{B}_S$.

**Theorem 5** *Let $\mathcal{B}_S$ and $\mathcal{B}_L$ be two disjoint sets of balls that roll on a plane where the balls in $\mathcal{B}_L$ are* *at least as large as the balls in $\mathcal{B}_S$. There is a KDS for collision detection between balls of $\mathcal{B}_S$ and balls of $\mathcal{B}_L$ that uses $O(|\mathcal{B}_S| + |\mathcal{B}_L|)$ space and processes $O((|\mathcal{B}_S| + |\mathcal{B}_L|)^2)$ events in the worst case if the balls follow low-degree algebraic trajectories. Each event can be handled in $O(1)$ time.*

## 3 Conclusions

This abstract describes a first step towards kinetic collision detection in 3 dimensions: a compact and responsive kinetic data structure for detecting collisions between $n$ balls of arbitrary sizes rolling on a plane. The full paper [1] presents additional results for convex fat 3-dimensional objects of constant complexity that are free-flying in $\mathbb{R}^3$. In that case we can detect collisions with a KDS of $O(n \log^6 n)$ size that can handle events in $O(\log^6 n)$ time. The structure processes $O(n^2)$ events in the worst case, assuming that the objects follow low-degree algebraic trajectories. If the objects have similar sizes then the size of the KDS becomes $O(n)$ and events can be handled in $O(1)$ time.

**Acknowledgements.** The last author would like to thank David Kirkpatrick for valuable discussions on the presented subject.

## References

[1] M. A. Abam, M. de Berg, S.-H. Poon, and B. Speckmann. Kinetic collision detection for convex fat objects. Submitted, 2005.

[2] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. *International Journal of Robotics Research*, 21:179–197, 2002.

[3] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection for two simple polygons. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 102–111, 1999.

[4] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31:1–28, 1999.

[5] J. Basch, L. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Symposium on Computational Geometry*, pages 344–351, 1997.

[6] J. Erickson, L. Guibas, J. Stolfi, and L. Zhang. Separation-sensitive collision detection for convex objects. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 327–336, 1999.

[7] L. Guibas. Kinetic data structures: A state of the art report. In *Proc. 3rd Workshop on Algorithmic Foundations of Robotics*, pages 191–209, 1998.

[8] L. Guibas. Motion. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1117–1134. CRC Press, 2nd edition, 2004.

[9] L. Guibas, F. Xie, and L. Zhang. Kineitc collision detection: Algorithms and experiments. In *Proc. International Conference on Robotics and Automation*, pages 2903–2910, 2001.

[10] D. Kim, L. Guibas, and S. Shin. Fast collision detection among multiple moving spheres. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):230–242, 1998.

[11] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry and Applications*, 12(1&2):3–27, 2002.

[12] D. Kirkpatrick and B. Speckmann. Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons. In *Proc. 18th ACM Symposium on Computational Geometry*, pages 179–188, 2002.

[13] M. Lin and D. Manocha. Collision and proximity queries. In J. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 787–807. CRC Press, 2nd edition, 2004.

# Computing Shortest Paths amidst Growing Discs in the Plane

Jur van den Berg[*] Mark Overmars[*]

## Abstract

In this paper an algorithm is presented to find a shortest path between two points in the plane amidst growing discs. That is, as the "point" moves through the plane, the discs grow at an a priori known rate. We present an $O(n^3 \log n)$ algorithm and a fast implementation. The problem is motivated from robotics, where motion planning in dynamic environments is a great challenge. Our algorithm can be used to generate paths in such environments guaranteeing that they will be collision-free in the future.

## 1 Introduction

An important challenge in robotics is motion planning in dynamic environments. That is, planning a path for a robot from a start location to a goal location that avoids collisions with the dynamic obstacles. In many cases the motions of the dynamic obstacles are not known beforehand, so their future trajectories are estimated by extrapolating current velocities (acquired by sensors) in order to plan a path [4].

A major problem is that the world is continuously changing: If some obstacles change their velocities (say at time $t$), a new trajectory should be planned. However, there actually is no time for this, no matter how fast it can be done, because at the time the calculation is finished the world has already changed, and hence the computation is outdated. To overcome this problem, often a fixed amount of time, say $\tau$, is reserved for planning. The planner then takes the expected situation of the world at time $t + \tau$ as initial world state, and the plan is executed when the time $t + \tau$ has come. This scheme carries two problems:

- The predicted situation of the world at time $t+\tau$ may differ from the actual situation when some obstacles change their velocities during planning. This may result in invalid paths.
- The path the robot will follow between time $t$ and time $t + \tau$ is not guaranteed to be collision free, because this path was computed based on the old velocities of the obstacles.

In this paper we describe a technique to overcome these problems. We present an algorithm that com-

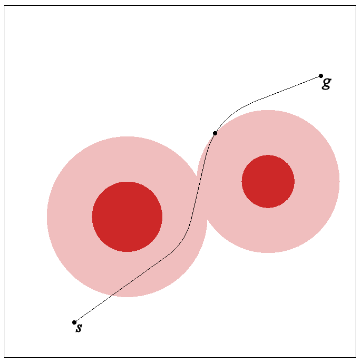


Figure 1: An environment with two dynamic obstacles and a shortest path. The dark and light discs depict the obstacles at $t = 0$ and $t = 1$ respectively. A small dot indicates the position along the path at $t = 1$.

putes a path from a start location to a goal location that is guaranteed to be collision-free, no matter how often the obstacles change their velocities in the future. Replanning might still be necessary from time to time, to generate trajectories with more appealing global characteristics, but the two problems identified above do not occur in this case. The first problem is solved by incorporating all the possible situations of the world at time $t + \tau$ in the world model. As the paths the algorithm computes are guaranteed to be collision-free no matter what the dynamic obstacles do, the second problem dissolves.

We assume that all obstacles and the robot are modeled as discs in the plane, and that the robot and the obstacles have a maximum velocity. The maximum velocity of the obstacles should not exceed the maximum velocity of the robot. The problem is solved in the configuration space, that is, the radius of the robot is added to the radii of the obstacles, so that we can treat the robot as a point.

Given the initial positions of each of the obstacles, we model the regions in which the obstacles might be as discs in the plane that grow over time. In this space, we compute a shortest path (a minimum time path) from a start configuration to a goal configuration that is collision-free with respect to the growing discs. Computing shortest paths is a well studied topic in computational geometry (see [5] for a survey). However, the problem posed in this paper presents some interesting challenges of its own. We present an algorithm that runs in $O(n^3 \log n)$ time, where $n$ is

[*]Department of Information and Computing Sciences, Universiteit Utrecht, berg@cs.uu.nl, markov@cs.uu.nl

the number of obstacles. Further, we created a fast implementation that generates shortest paths at interactive rates.

## 2 Problem definition

The problem is formally defined as follows. Given are $n$ dynamic obstacles $O_1, \ldots, O_n$ which are discs in the plane. The centers of the discs at time $t = 0$ are given by the coordinates $p_1, \ldots, p_n \in \mathbb{R}^2$, and the radii of the discs by $r_1, \ldots, r_n \in \mathbb{R}^+$. All of the obstacles have the same maximal velocity, given by $v \in \mathbb{R}^+$. The robot is a point (if it is a disc, it can be treated as a point when its radius is added to the radii of the obstacles), for which a path should be found between a start configuration $s \in \mathbb{R}^2$ and a goal configuration $g \in \mathbb{R}^2$. The robot has a maximal velocity $V \in \mathbb{R}^+$ which should be larger than the maximal velocity of the obstacles, i.e. $V > v$.

As we do not assume any knowledge of the velocities of the dynamic obstacles, other than that they have a maximal velocity, the region that is guaranteed to contain all the dynamic obstacles at some point in time $t$ is bounded by $\bigcup_i B(p_i, r_i+vt)$, where $B(p, r) \subset \mathbb{R}^2$ is an open disc centered at $p$ with radius $r$. In other words, each of the dynamic obstacles is conservatively modeled by a disc that grows over time with a rate corresponding to its maximal velocity (see Fig. 1 for an example environment).

**Definition 1** *A point $p \in \mathbb{R}^2$ is* collision-free *at time $t \in \mathbb{R}^+$ if $p \notin \bigcup_i B(p_i, r_i + vt)$.*

The goal of the problem is to compute the shortest possible path $\pi : [0, T] \to \mathbb{R}^2$ between $s$ and $g$ (i.e. a minimal time path) that is collision-free with respect to the growing discs for all $t \in [0, T]$.

## 3 Properties

**Observation 1** *A point $p \in \mathbb{R}^2$ that is collision-free at time $t = t'$, is collision-free for all $t :: 0 \leq t \leq t'$.*

**Theorem 1** *The velocity $\frac{||(\delta x, \delta y)||}{\delta t}$ of a shortest path is constant and equal to the maximal velocity $V$.*

**Proof.** Suppose $\pi$ is a path to $g$, of which a sub-path has a velocity smaller than $V$. Then this sub-path could have been traversed at maximal velocity, so that points further along the path would be reached at an earlier time. Observation 1 proves that these points are then collision-free as well, so also $g$ could have been reached sooner, and hence $\pi$ is not a shortest path. $\square$

**Theorem 2** *A shortest path consists only of straight line segments, and segments of a* logarithmic spiral *incident to the boundary of a growing disc.*

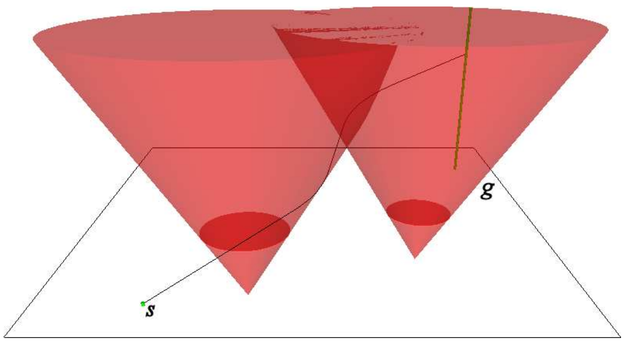


Figure 2: The three-dimensional space of the same environment as Fig. 1.

**Proof.** Theorem 1 implies that the time it takes to traverse a path is proportional to its length. Hence, parts of the path in 'open' space can always be shortcut by a straight-line segment. Only when the path stays incident to the boundary of a growing disc, it is not possible to shortcut. As both the velocity of the path, and the growth rate of the disc are constant, it is easily shown that such segments are part of a logarithmic spiral [6]. $\square$

**Corollary 3** *A shortest path is $C^1$-smooth.*

**Proof.** Suppose $\pi$ is a path containing sharp turns. Then these turns could be shortcut by a straight-line segment, and hence $\pi$ is not a shortest path. Thus, in a shortest path the straight-line segments are tangent to the supporting spirals of the spiral segments. $\square$

## 4 Global Approach

As the discs grow over time, we can see the obstacles as cones in a three-dimensional space (Fig. 2), where the third dimension models the time. Each obstacle $O_i$ transforms into a cone, whose central axis is parallel to the time-axis of the coordinate frame, and intersects the $xy$-plane at point $p_i$. The maximal velocity $v$ determines the opening angle of the cone, and the initial radius $r_i$ determines the (negative) time-coordinate of the apex. The goal configuration is transformed into a line parallel to the time-axis, where we want to arrive as soon as possible (i.e. for the lowest value of $t$). In this space it is easier to reason about the algorithm we devise to find a shortest path.

Our algorithm to solve the problem is based on a Dijkstra's shortest-path search [3], which starts from the start configuration $s$. By Theorem 1, all straight-line segments emanating from $s$ of which the slope equals the maximal velocity $V$, are possible initial motions. This set is narrowed down by Corollary 3, which implies that only the segment leading directly to the goal, and straight-line segments tangent to the cones are possibly part of the shortest path to the goal. These segments may intersect other cones, which would make them invalid, so only the collision-free segments are considered. Each of them is put into a priority queue $\mathcal{Q}$ with a key corresponding to the $t$-value of its endpoint.

Now, the algorithm proceeds by handling the point with the lowest $t$-value in the queue (the front element of $\mathcal{Q}$). This point is either the goal location, in which case the shortest path has been found, or a point on the surface of a cone. In this latter, more general case we proceed similarly by finding straight-line segments tangent to other cones and to the goal configuration. However, as we are on the surface of a cone, we first have to walk a piece of a spiral around the cone such that the straight-line segment is tangent to both the 'source' cone and the 'destination' cone. For each cone, as well as for the goal configuration, these segments are computed and if collision-free their endpoints are inserted in $\mathcal{Q}$.

This procedure is repeated until the goal configuration is popped from the priority queue. In this case the shortest path has been found, and can be read out if backpointers have been maintained during the algorithm. If the priority queue becomes empty, no valid path exists.

## 5 Details

The algorithm described above will indeed find a shortest path to the goal. However, in order to have a finite bound on the running time we must define 'nodes' that can provably be visited only once in a shortest path, such that we can do *relaxation* on them as in Dijkstra's algorithm.

Let us look at the following. A shortest path consists of spiral segments on a cone's surface and straight-line segments that are bitangent to two cones. There are four ways in which a segment can be bitangent to a pair of cones: left-left, right-right, left-right and right-left. In each of these cases, there is an infinite number of bitangent segments with a slope corresponding to the maximal velocity $V$, but the possible tangency points at the source cone form a continuous curve on the surface of the cone. We call such curve a *departure curve.*

The departure curve may be cut into several collision-free *intervals* by other cones that penetrate the surface of the cone. These intervals form the 'nodes' in our search process. Only the path arriving earliest in an interval can contribute to a shortest path. Other paths arriving later in the interval cannot be part of the shortest path, because the path arriving earliest in the interval can be extended with a traversal along the interval to end up at the same position (and time) as the path arriving later in the interval. This argument applies if for the departure

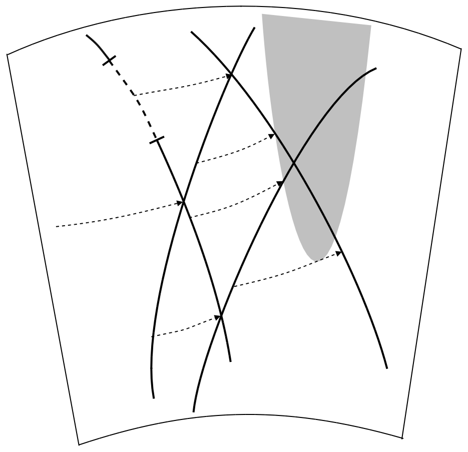


Figure 3: An impression of an arrangement on the surface of the cone. The thick lines are the departure curves, of which one has a shadow interval (dashed). The thin dashed lines are spiral segments that delimit trapezoidal regions that have the same next departure curve or collision (only the counter-clockwise spirals are shown). The gray area depicts an intersection area of another cone penetrating the surface, and cutting several departure curves into two intervals.

curve holds that $\frac{||(\delta x, \delta y)||}{\delta t} \leq V$. This is only the case when all cones have the same opening angle. (This explains our assumption that all discs have the same maximal velocity $v$.) As the proof is rather technical, we omit it here.

To identify these departure curve intervals, we compute an *arrangement* [1] on the surface of each cone of all departure curves on that cone (see Fig. 3). Areas on the surface on the cone that are intersected by other cones are also inserted in this arrangement. The intervals can then be extracted from the arrangement.

Each departure curve interval has two outgoing edges. One –a spiral segment– to the next departure curve on the source cone, and one –consisting of a bitangent straight-line segment and a spiral segment– to the first departure curve encountered on the destination cone that is associated with the departure curve. For the first edge, which stays on the cone, we have to determine the next departure curve that is encountered if we proceed by moving along the spiral around the cone. This can be done using the arrangement, if we have computed its *trapezoidal map* [1], where the sides of the trapezoids are spiral segments.

For the second edge, which crosses to another cone, we have to determine what the first departure curve is we will encounter there. This is done using the arrangement we have computed on that cone. Using a point-location query, we can determine in what cell of the arrangement the straight-line segment has arrived, and using the trapezoidal map we know what the first departure curve is we will encounter if we proceed from there.

Finally, we must ascertain that each edge is collision-free with respect to the other cones. Spiral segments may collide with other cones if these penetrate the spiral's cone surface. Since intersection areas are incorporated into the arrangement, such collisions are easily detected. Straight-line segments may collide with any cone, so for each departure curve and each cone, we calculate the 'shadow' interval this cone casts on the departure curve, in which a departure will result in collision. These shadow intervals are stored in the arrangement as well. In Fig. 3, an impression is given of how such an arrangement might look.

**Theorem 4** *The algorithm to compute a shortest path amidst $n$ growing discs runs in $O(n^3 \log n)$ time.*

**Proof.** For each pair of cones there are $O(1)$ departure curves. Since there are $O(n^2)$ pairs of cones, there are $O(n^2)$ departure curves in total. Each of the departure curves can be segmented into at most $O(n)$ intervals, as there are $O(n)$ cones possibly intersecting the departure curve. (Each cone can split the departure curve in at most two segments.) Hence, there are $O(n^3)$ departure curve intervals. Each departure curve interval has $O(1)$ outgoing edges, making a total of $O(n^3)$ edges.

The complexity of Dijkstra's algorithm is known to be $O(N \log N + E)$ where $N$ is the number of nodes, and $E$ the number of edges. Each edge requires some additional work. Firstly, we have to find the departure curve interval in which it will arrive, by doing a point-location query in the trapezoidal map of one of the arrangements. This takes $O(\log n)$ time. Further, we must determine whether an edge is collision-free. Using the shadow intervals stored at the departure curves, this can be done in $O(\log n)$ time as well. Thus, as both $N$ and $E$ are $O(n^3)$, Dijkstra's algorithm will run in $O(n^3 \log n)$ time in total.

Computing the arrangements and their trapezoidal maps takes $O(n^2)$ time per cone, as there are $O(n)$ departure curves on each cone, and $O(n)$ intersection areas of other cones. As there are $O(n)$ cones, this step takes $O(n^3)$ time in total. All the shadow intervals can be computed in $O(n^3)$ time as well, as there are $O(n^2)$ departure curves and $O(n)$ cones.

Overall, we can conclude that our algorithm runs in $O(n^3 \log n)$ time. □

## 6 Implementation

We created a fast implementation of the algorithm presented above. Instead of using the arrangements, we used some ad-hoc approaches for doing the elementary tests. These may have a slower asymptotic running time, but in practice they turned out to be fast. For example, intersections between a spiral and a departure curve cannot be found analytically, so we used a combination of two approximate root-finding algorithms [2]. The Dijkstra method was replaced by an equally suited A*-method [4], that is faster in practice as it focusses the search to the goal. The implementation runs at interactive rates even for many obstacles. For example, a shortest path among 10 cones is computed within 0.01 seconds on a Pentium IV 3.0GHz with 1 GByte of memory. Figs. 1 and 2 were created using our implementation.

## 7 Conclusion

In this paper we presented an algorithm for computing shortest paths (minimum time paths) amidst discs that grow over time. A growing disc could model the region that is guaranteed to contain a dynamic obstacle of which the maximal velocity is given. Hence, using our algorithm, paths can be found that are guaranteed to be collision-free in the future, regardless of the behavior of the dynamic obstacles. As the regions grow fast over time, a new path should be planned from time to time –based on newly acquired sensor data– to generate paths with more appealing global characteristics. Our implementation shows that such paths can be generated quickly. A great advantage over other methods is that this replanning can be done safely. The old path that is still used *during* replanning is guaranteed to be collision-free. A requirement though, is that the "robot" has a higher maximal velocity than any of the dynamic obstacles.

A drawback of the method we presented is that a path to the goal often does not exist. This occurs when the goal is covered by a growing disc before it can be reached. A solution to this problem would be to find the path that comes closest to the goal. It seems that this can easily be incorporated in our algorithm, but it is still subject of ongoing research.

## References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry, Algorithms and Applications. Chapters 6 and 8. Springer-Verlag, Berlin Heidelberg, 1997.

[2] R. L. Burden, J. D. Faires. Numerical analysis, 7th edition. Chapter 2. Brooks/Cole, Pacific Grove, 2001.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269-271, 1959.

[4] S. M. LaValle. Planning Algorithms. Cambridge University Press, 2006.

[5] J. S. B. Mitchell. Geometric shortest paths and network optimization. In Handbook of Computational Geometry, pages 633-701. Elsevier Science Publishers, Amsterdam, 2000.

[6] E. W. Weisstein. Logarithmic Spiral. In MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/LogarithmicSpiral.html

# Few Optimal Foldings of HP Protein Chains on Various Lattices*

Sheung-Hung Poon† Shripad Thite†

## Abstract

We consider whether or not protein chains in the HP model have unique or few optimal foldings. We solve the conjecture proposed by Aichholzer et al. that the open chain $\mathcal{L}_{2k-1} = (HP)^k(PH)^{k-1}$ for $k \geq 3$ has exactly two optimal foldings on the square lattice. We show that some closed and open chains have unique optimal foldings on the hexagonal and triangular lattices, respectively.

## 1 Introduction

Protein folding is a central and long-standing problem in molecular and computational biology. Due to the complexity of the problem, a variety of simplified models have been proposed to simulate how real proteins fold. In the Hydrophobic-Polar (HP) model, the amino acids in proteins are grouped into two types: *hydrophobic* ($H$) monomers and *hydrophilic* or *polar* ($P$) monomers. $H$ monomers tend to attract each other while $P$ monomers are neutral. Proteins are modeled as chains of $H$ and $P$ nodes, or equivalently, strings from $\{H, P\}^+$. The chains are embedded in some lattice in two or three dimensions such that monomers which are adjacent in the given chain must be placed at adjacent points in the lattice. Two non-adjacent nodes on the chain are in *contact* if they occupy a pair of neighboring lattice points. An *optimal folding* of a chain is an embedding in the lattice which maximizes the number of $HH$ contacts.

Much research has been done on the HP model. In particular, Berger and Lieghton [2] showed the NP-completeness of finding the optimal folding on the cubic lattice in 3D, and Crescenzi et al. [3] proved the NP-completeness on the square lattice in 2D. Constant-factor approximation algorithms were also developed for various lattices in both 2D and 3D. We consider the question of whether or not chains in HP model have unique or few optimal foldings. The problem is related to the folding stability of protein chains, and was first suggested by Hayes [4]. Aichholzer et al. [1] exhibited families of closed and open chains in the square lattice, each of which has a unique optimal folding. In this paper, we obtain several results for the square, hexagonal and triangular lattices in two dimensions.

*S.-H.P. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 612.065.307. S.T. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

†Department of Mathematics and Computer Science, TU Eindhoven, 5600 MB, Eindhoven, the Netherlands. {spoon,sthite}@win.tue.nl

## 2 Open Chain in Square Lattice

Consider the open chain $\mathcal{L}_{2k-1} = (HP)^k(PH)^{k-1}$. In this section, we solve a conjecture proposed by Aichholzer et al. [1] by showing the theorem below.

**Theorem 1** *The open chain $\mathcal{L}_{2k-1}$ for $k \geq 3$ has exactly two optimal foldings on the square lattice.*

First, we need the theorem from [1] about unique optimal folding of the closed chain as stated below. See Figure 1 for examples. Note that, in our figures, we use small circles to denote $H$ nodes and small black disks to denote $P$ nodes; we use solid segments to denote chain edges and dashed ones to denote $HH$ contacts.

**Theorem 2** *[1] The closed chain $\mathcal{S}_k = P(HP)^{\lceil k/2\rceil}P(HP)^{\lfloor k/2\rfloor}$ for $k \geq 1$ has a unique optimal folding on the square lattice.*

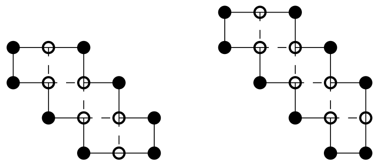


Figure 1: Optimal foldings of $\mathcal{S}_6$ and $\mathcal{S}_7$.

Aichholzer et al. [1] show that Fact 18 to Lemma 29 in their paper hold for the open chain $\mathcal{L}_{2k} = (HP)^k(PH)^k$ for $k \geq 1$. We can verify that these properties are also true for $\mathcal{L}_{2k-1}$. However, for the later lemmas and theorems in their paper, adjustments need to be made to be suitable for the chain $\mathcal{L}_{2k-1}$. The two lemmas below simulate Lemmas 30 and 31 in [1], and their proofs can be adapted with slight modifications. A *straight* node is a node collinear with both its preceding and following nodes on the chain. A *solitary straight H node* $v$ is a straight

$H$ node on the bounding box $B$ of the chain such that both its preceding and following $H$ nodes are not on the same side of $B$ as $v$.

**Lemma 3** *In an optimal folding of $\mathcal{L}_{2k-1}$, there are either one or two solitary straight $H$ nodes on its bounding box $B$. In particular, if there are exactly two solitary straight $H$ nodes on $B$, then (see Figure 2(a))*

*(i) They lie on opposite sides of $B$.*

*(ii) One of them is adjacent to the $PP$ edge, and the other is adjacent to an end edge $uv$ and in contact with an endpoint.*

*(iii) The $PP$ edge and the end edge $uv$ lie on opposite sides of $B$.*

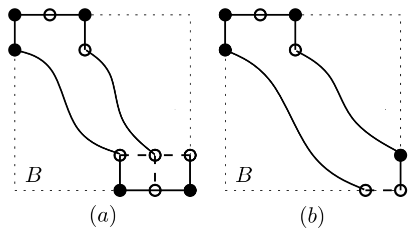


Figure 2: Optimal foldings: $(a)$ when there are two solitary straight $H$ nodes; $(b)$ when there are only one.

**Lemma 4** *In an optimal folding of $\mathcal{L}_{2k-1}$, if there is exactly one solitary straight $H$ node on its bounding box $B$, then (see Figure 2(b))*

*(i) The solitary $H$ node is adjacent to the $PP$ edge.*

*(ii) The solitary $H$ node and the contact of the two endpoints of the chain lie on opposite sides of $B$.*

*(iii) The $PP$ edge and an end edge of the chain lie on opposite sides of $B$.*

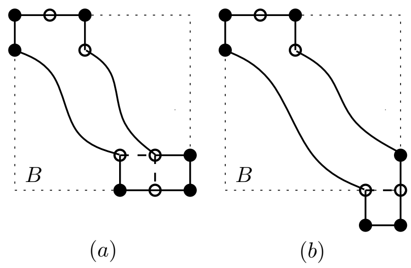


Figure 3: Modify cases $(a)$ and $(b)$ in Figure 2 to closed chains $\mathcal{S}_{2k-2}$ and $\mathcal{S}_{2k-1}$ respectively.

Now we are ready to prove our main theorem.

**Proof of Theorem 1.** Case $(a)$: If there are exactly two solitary $H$ nodes, by Lemma 3 we can modify the optimal folding of $\mathcal{L}_{2k-1}$ to an optimal folding of $\mathcal{S}_{2k-2}$ by adding a chain edge between the contact of the two end nodes and replacing the end $H$ node on the chain bounding box to a $P$ node. See Figure 3 $(a)$. Thus in this case, the number of optimal folding(s) of $\mathcal{L}_{2k-1}$ is equal to that of $\mathcal{S}_{2k-2}$, which is one by Theorem 2.

Case $(b)$: If there is exactly one solitary $H$ node, by Lemma 4 we can modify the optimal folding of $\mathcal{L}_{2k-1}$ to an optimal folding of $\mathcal{S}_{2k-1}$ by connecting the two end $H$ nodes by a short chain $HPPH$. See Figure 3 $(b)$. Thus in this case, the number of optimal folding(s) of $\mathcal{L}_{2k-1}$ is equal to that of $\mathcal{S}_{2k-1}$, which is one by Theorem 2. □

## 3 Hexagonal Lattice

### 3.1 Closed chain

Consider the closed chain $\mathcal{H}_k = (HP)^k PPP(HP)^k PPP$ for $k \geq 1$. We call the two subchains $PPPP$ the two *ends* of $\mathcal{H}_k$. In the above expression of $\mathcal{H}_k$, we denote the $i$th $H$ node by $H_i$ for $1 \leq i \leq 2k$. We consider the folding $\mathcal{F}_k$, in which each $H_i$ for $1 \leq i \leq k$ is in contact with $H_{2k-i+1}$. See Figure 4 for an example of folding $\mathcal{F}_k$. We call a contact between an $H$ node and a non-$H$ node a *missing contact.*

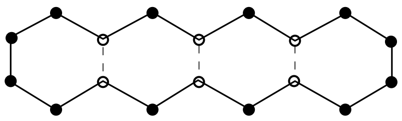


Figure 4: Folding $\mathcal{F}_3$ for $\mathcal{H}_3$.

As in folding $\mathcal{F}_k$, all $H$ nodes are in contact with other $H$ nodes. As there is no missing contact in $\mathcal{F}_k$, there is also none in the optimal folding. Now suppose each $H_i$ for $1 \leq i \leq k$ is in contact with $H_{c_i}$ in the optimal folding. Due to the parity of the positions of $H$ nodes, we have $c_i > k$. We claim that $c_i$ decreases as $i$ increases in the lemma below. After we have the claim, our theorem is immediate.

**Lemma 5** *Suppose each $H_i$ for $1 \leq i \leq k$ is in contact with $H_{c_i}$ in the optimal folding. Then $c_i$ decreases as $i$ increases.*

**Proof.** Suppose to the contrary that there exist $i, i'(i < i')$ such that $c_i < c_{i'}$. Note that $H_i$ (resp. $H_{i'}$) is in contact with $H_{c_i}$ (resp. $H_{c_{i'}}$). Denote the subchain from $H_i$ to $H_{i'}$ (resp. from $H_{c_i}$ to $H_{c_{i'}}$) not containing any end of $\mathcal{H}_k$ by $C_1$ (resp. $C_2$). Denote the subchain from $H_i$ to $H_{c_{i'}}$ containing one end of $\mathcal{H}_k$ by $E_1$. And also denote the subchain from $H_{i'}$ to

$H_{c_i}$ containing another end of $\mathcal{H}_k$ by $E_2$. See Figure 5 for illustration.

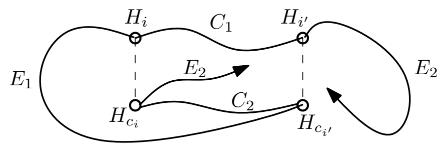


Figure 5: Illustration for the proof of Lemma 5.

Note that there are no chain edges or contacts that can intersect the contact $H_{i'}H_{c_{i'}}$. Consider the cycle $D = C \cup E_1 \cup H_{i'}H_{c_{i'}}$. As $H_i$ is in contact with $H_{c_i}$, it is not hard to see that $H_{c_i}$ must be in the interior of cycle $D$. Also it is clear that $E_2$ lies in the exterior of cycle $D$. As $E_2$ connects $H_{i'}$ and $H_{c_i}$, $E_2$ must intersect the contact $H_{i'}H_{c_{i'}}$. This is a contradiction. □

**Theorem 6** *The closed chain $\mathcal{H}_k$ for $k \geq 1$ has the unique optimal folding $\mathcal{F}_k$ on the hexagonal lattice.*

**Proof.** By Lemma 5, $c_i$ decreases as $i$ increases from 1 to $k$ in any optimal folding. As all $c_i$ are different and $c_i \in \{k+1, \ldots, 2k\}$, $c_i$ must be $2k - i + 1$. Thus $\mathcal{F}_k$ is the unique optimal folding. □

## 3.2 Open chain

Consider the open chain $\mathcal{H}'_k = P(HP)^k PPP(HP)^k$ for $k \geq 1$. In the above expression, we denote the $i$th $H$ node by $H_i$ for $1 \leq i \leq 2k$. We consider the folding $\mathcal{F}'_k$, in which each $H_i$ for $1 \leq i \leq k$ is in contact with $H_{2k-i+1}$. Notice that $\mathcal{F}'_k$ simulates $\mathcal{F}_k$. See Figure 6 for an example of $\mathcal{F}'_k$.

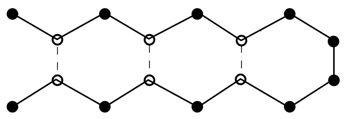


Figure 6: Folding $\mathcal{F}'_3$ for $\mathcal{H}'_3$.

The uniqueness of the optimal folding for $\mathcal{H}'_k$ can be shown by following the similar proof skeleton as Theorem 6, but with slightly more involved arguments.

**Theorem 7** *The open chain $\mathcal{H}'_k$ for $k \geq 1$ has the unique optimal folding $\mathcal{F}'_k$ on the hexagonal lattice.*

## 4 Triangular Lattice

### 4.1 Closed chain

Consider the closed chain $\mathcal{T}_k = (HP)^k$. We consider its folding $\mathcal{G}_k$ defined as shown in Figure 7.

In this section, we show the following uniqueness theorem. Note that the theorem is not true for $k = 6$.

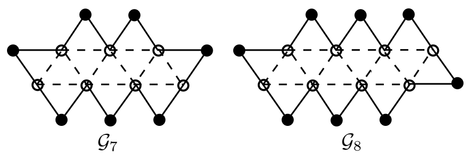


Figure 7: Foldings $\mathcal{G}_7$ & $\mathcal{G}_8$ for $\mathcal{T}_7$ & $\mathcal{T}_8$ respectively.

**Theorem 8** *The closed triangular chain $\mathcal{T}_k$ for $k \geq 2$ and $k \neq 6$ has the unique optimal folding $\mathcal{G}_k$ on the triangular lattice.*

When $k$ is small, we can show the uniqueness of the optimal folding by enumerating the configurations of the $HH$-contact graph with maximum number of contacts.

**Lemma 9** *The chain $\mathcal{T}_k$ for $2 \leq k \leq 5$ or $k = 7$ has the unique optimal folding $\mathcal{G}_k$. The chain $\mathcal{T}_6$ has two optimal foldings including $\mathcal{G}_k$ as shown in Figure 8.*

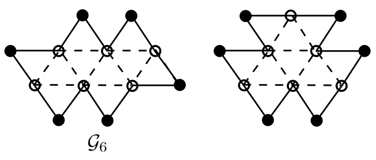


Figure 8: Two optimal foldings of $\mathcal{T}_6$.

It remains to show the uniqueness of the optimal folding of long chains as stated in the following main lemma.

**Lemma 10** *The chain $\mathcal{T}_k$ for $k \geq 8$ has the unique optimal folding $\mathcal{G}_k$.*

As there are six missing contacts in $\mathcal{G}_k$, we observe that an optimal folding has at most six missing contacts.

We call an $H$ node *fully-contacted* if there is no missing contact from it. The optimal folding of $\mathcal{T}_k$ for $k \geq 8$ contains at least two fully-contacted $H$ nodes due to the above observation. By careful examination of the neighborhoods of the two $H$ nodes, we can show that there must be a pair of contacting $H$ nodes that are both fully-contacted and non-straight.

**Lemma 11** *An optimal folding of $\mathcal{T}_k$ for $k \geq 8$ contains two fully-contacted non-straight H nodes in contact with each other.*

Using the above lemma, we can divide the whole chain at a pair of contacting $H$ nodes into two "quite-long" paths.

**Lemma 12** *An optimal folding of $\mathcal{T}_k$ for $k \geq 8$ contains two non-straight contacting H nodes such that they divide $\mathcal{T}_k$ into two paths, each of which contains at least two internal H nodes.*

We define a *U-line* (resp. *D-line*) as a line of slope $\sqrt{3}$ (resp. $-\sqrt{3}$). We define a *canonical line* of the triangular lattice as a horizontal line, a U-line, or a D-line. A *canonical strip* of a lattice edge $e$ in the triangular lattice is a strip between the two parallel canonical lines, each of which passes through exactly one endpoint of $e$. Note that each lattice edge has exactly two canonical strips.

**Lemma 13** *Suppose $\mathcal{C}$ is a path along $\mathcal{T}_k$ connecting a pair of contacting H nodes such that $\mathcal{C}$ contains either a non-straight internal H node or two internal H nodes. Then there are at least three missing contacts from internal H nodes of $\mathcal{C}$.*

**Proof.** (*Sketch*) Suppose $X$ is a canonical strip of the contacting edge $e$ between the pair of ending $H$ nodes such that the two end edges of $\mathcal{C}$ are separated by $X$. Without loss of generality, we assume that $X$ runs horizontally, the contact edge $e$ between the two end $H$ nodes of $\mathcal{C}$ lies on a U-line, and $\mathcal{C}$ crosses $X$ to the right of $e$ in an odd number of times. See Figure 9 for illustration. Let $H_a, H_b$ be the upper and lower ends of $e$ respectively.

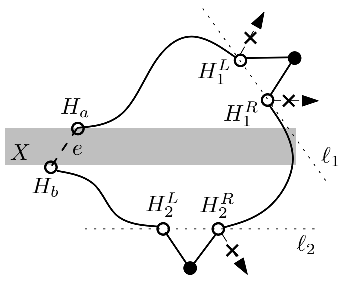


Figure 9: Illustration for the proof of Lemma 13.

Sweep a D-line to the right until it reaches some extremal $H$ node of $\mathcal{C}$. We call the D-line at current position $\ell_1$. Let $H_1^L$ and $H_1^R$ be the leftmost and rightmost $H$ nodes on $\ell_1$ respectively. We define $\ell_2, H_2^L, H_2^R$ similarly by sweeping a horizontal line downwards.

It is clear that the right-contact of $H_1^R$ and the bottom-right-contact of $H_2^R$ are both missing. With the given conditions, it is easy to show that $H_1^L = H_a$ and $H_2^L = H_b$ cannot both be true. Without loss of generality, we assume that the former is not true. Then we have that the top-right-contact of $H_1^L$ is also missing. □

Now by an involved analysis, we can show that in order for each of these two paths to contain exactly three missing contacts, it must possess the pattern as shown in Figure 10 ($a$) or ($b$). The details are omitted in this abstract. With this property, it is immediate to claim our main lemma, Lemma 10, and we finish the proof of Theorem 8.

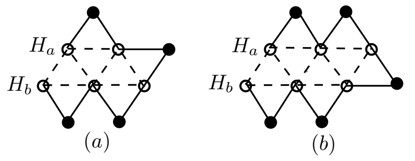


Figure 10: Patterns in an optimal folding.

## 4.2 Open chain

However, the open chain $\mathcal{T}'_k = (HP)^{k-1}H$ can have several optimal foldings on the triangular lattice. Instead, we show the following theorem for the open chain $\mathcal{T}''_k = (HP)^k(PHP)^2(PH)^k$ for $k \geq 3$ by using the similar technique we use for the closed chain $\mathcal{T}_k$, but with a more involved analysis. See Figure 11 for an example of the unique optimal folding.

**Theorem 14** *The open chain $\mathcal{T}''_k$ for $k \geq 3$ has a unique optimal folding on the triangular lattice.*

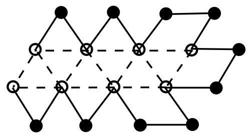


Figure 11: The unique optimal folding of $\mathcal{T}''_3$.

## 5 Conclusion & Discussion

We solve a conjecture about an open chain in the square lattice. We obtain unique optimal foldings for chains in the hexagonal and triangular lattices, respectively. All of our results are in two dimensions. Is there any family of chains that have unique optimal foldings on some lattice in three dimensions?

## References

[1] O. Aichholzer, D. Bremner, E. Demaine, H. Meijer, V. Sacristan, and M. Soss. Long proteins with unique optimal foldings in the H-P model. *Computational Geometry: Theory and Applications*, 25(1-2), 139–159, 2003.

[2] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1), 27–40, 1998.

[3] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3), 423–466, 1998.

[4] B. Hayes. Prototeins. *American Scientist*, 86, 216–221, 1998.

# Reconfiguring planar dihedral chains

Greg Aloupis [*] Henk Meijer [†]

## Abstract

We consider the dihedral model of motion for chains with fixed edge lengths, in which the angle between every pair of successive edges remains fixed. A chain is *flat-state connected* if every planar configuration can be transformed to any other via a series of dihedral motions which maintain simplicity. Here we prove that three classes of chains are flat-state connected. The first class is that of chains with unit-length edges and all angles in the range $(60°, 150°)$. The second is the class of chains for which a planar monotone configuration exists. The third class includes, but is not limited to, chains for which every angle is in the range $(\delta, 2\delta)$, for $\delta \leq \frac{\pi}{3}$.

## 1 Introduction

The dihedral model for three-dimensional linkages resembles (and is designed to approximate) the "ball and stick" molecular model, used in introductory chemistry courses. Edges have fixed lengths and are not allowed to intersect. The angle between any two edges with a common vertex must also remain fixed. Thus any linkage motion can be decomposed into basic dihedral motions. A basic dihedral motion is defined on a selected edge of a given linkage. The entire linkage on one side of the selected edge is rotated rigidly about the axis of the edge (see Figure 1). This maintains all angles fixed.

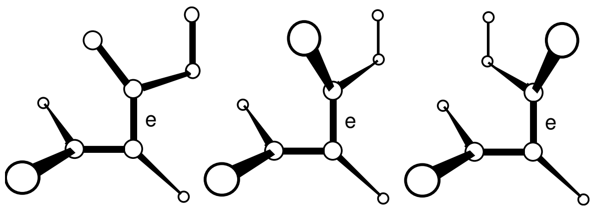


Figure 1: Three snapshots of a dihedral motion about edge $e$.

Soss and Toussaint [6] showed that deciding whether a chain can be flattened is NP-hard. They also developed a quadratic time algorithm to determine if the dihedral rotation about one edge results in edge-crossings. They gave a lower bound of $\Omega(n \log n)$, and this was nearly matched in the special case where the rotation is a full revolution. Soss, Erickson and Overmars [4] showed that pre-processing hardly helps if a series of such rotation queries is to be made. Several results on dihedral reconfigurations appear in the doctoral thesis by Soss [5]. A main open problem remaining from that research is to determine whether all planar (flat) configurations of a given chain are connected by a series of dihedral motions. This has come to be known as the *flat-state connectivity problem*, and accordingly when the answer is positive we say that a chain is "flat-state connected". One reason that planar configurations have received attention is that they may be useful as an intermediate (canonical) form during the reconfiguration of three-dimensional linkages. Demaine, Langerman and O'Rourke [3] considered chains with non-acute angles that can be "produced" through the apex of a cone, in a first geometric attempt to model a "protein machine". It was shown that chains are producible if and only if they can be flattened. Problems regarding flat-state connectivity were solved in [1, 2], by imposing additional restrictions to polygons, chains and trees. For example, orthogonal graphs were shown to be flat-state *disconnected*, as well as other linkages which were partially rigid. Open chains with non-acute or equal angles are flat-state connected, as are orthogonal polygons with unit edges. Of particular relevance to this paper is the class of open chains with unit-length edges and all angles in the range $(60°, 90°)$, shown to be flat-state connected in [2]. Here, we significantly expand this range of angles and prove that two new classes of open chains are flat-state connected.

In the following sections, we say that a chain consists of vertices $v_0, \ldots, v_n$ and edges $e_0, \ldots, e_{n-1}$. The edge $e_i$ connects vertices $v_i$ and $v_{i+1}$. Let $\alpha_i$ denote the angle at $v_i$, between $e_{i-1}$ and $e_i$ [1].

## 2 Chains which have a monotone configuration

**Theorem 1** *Open chains that have a strictly monotone flat embedding are flat-state connected.*

[*]Département d'Informatique, Université Libre de Bruxelles, greg.aloupis@ulb.ac.be

[†]School of Computing, Queen's University, henk@cs.queensu.ca

[1]Not to be mistaken with the *turning angle* of the chain. Here we have $(0 < \alpha_i < \pi)$. $\alpha_i = \pi$ would mean that the two edges could be considered as one and $v_i$ could be ignored.

**Proof.** (sketch) Let $D$ be a flat strictly monotone embedding of an open chain, i.e. any line parallel to the $y$-axis intersects $D$ at a single point. Let $C$ be another flat embedding of the same chain. We will prove that $C$ can be reconfigured to $D$ using dihedral motions. This implies the theorem, since $D$ serves as a canonical configuration.

Let $C$ be embedded in some plane $Q$. Let $l$ be a line that lies in $Q$. Let $P$ be a half plane bounded by $l$, contained above $Q$, as shown in Figure 2(a).

At the start of iteration $i$, $e_0, e_1, \ldots, e_{i-1}$ lie in $P$ and conform to the layout of these edges in $D$, i.e. any line in $P$ parallel to $l$ intersects $D$ at most once. The edges $e_i, e_{i+1}, \ldots, e_{n-1}$ lie in $Q$ and these edges have the layout of $C$. The angle between $P$ and $Q$ is almost zero. Either $e_i$ has to rotate around the edge $e_{i-1}$ by an angle of almost $\pi$, or it is already nearly in the correct position. In the latter case all we need to do is lift $v_i$, $l$ and $P$ a little out of $Q$ until $e_i$ lies in the plane containing $P$. The edge $e_i$ and the chain in $Q$ have to be rotated slightly for this motion to be possible.

So assume that $e_i$ is not in its correct position . Let $e'_i$ denote the location where it has to move to. There are three cases to be considered. Either $e_i$ and $e'_i$ lie on the same side of $l$, they lie on opposite sides of $l$, or $e_i$ lies on $l$. Since $D$ is strictly monotone $e'_i$ cannot lie on $l$. The first two cases are illustrated in Figure 2(b) and (c).

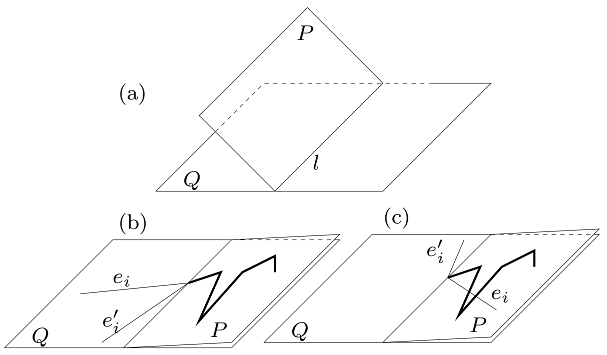


Figure 2: (a) Planes $P$ and $Q$; (b) $e_i$ and $e'_i$ on the same side of $l$; (c) $e_i$ and $e'_i$ on opposite sides $l$

The theorem can be proved by examining the possible values of $\alpha_i$. Let $\phi$ be the smallest angle between $e_{i-1}$ and $l$. Let $\theta$ be the smallest angle between $e_{i-1}$ and plane $Q$ so $0 < \theta \leq \phi \leq \pi/2$.

We first observe that $\alpha_i > \phi$ since $D$ is strictly monotone. If $e_i$ and $e'_i$ lie on the same side of $l$ we have $\alpha_i > \pi - \phi$. This condition implies that $\alpha_i > \pi/2$. We rotate $P$ around $l$ until $\theta$ is equal to $\pi - \alpha_i$. This happens before $P$ is vertical. During the rotation we have $0 < \theta \leq \pi - \alpha_i < \phi$. For each value of $\theta$ we can rotate $C_0$ around $v_i$ so that the angle between $e_{i-1}$ and $e_i$ falls in the range $[\theta, \pi - \theta]$. This range contains $\alpha_i$ since $\alpha_i \leq \pi - \theta$ and $\theta < \pi/2 < \alpha_i$. So we can maintain an angle of $\alpha_i$ between $e_{i-1}$ and $e_i$ by rotating $C_0$ in plane $Q$ around $v_i$. Similarly we can show that the layout after iteration $i$ can be moved to the same intermediate configuration. This is sufficient to prove this case.

If $e_i$ and $e'_i$ lie on different sides of $l$ we have $\alpha_i < \pi - \phi$. We will show that we can rotate $P$, while $e_i$ stays on the same side of $l$. We first increase the angle between $P$ and plane $Q$ from $\epsilon$ to $\pi/2$. The range $[\theta, \pi - \theta]$ contains $\alpha_i$ since $\alpha_i < \pi - \phi \leq \pi - \theta$. So we can maintain an angle of $\alpha_i$ between $e_{i-1}$ and $e_i$. Since the largest angle between $l$ and $e_{i-1}$ remains constant at value $\pi - \phi$ and is greater than $\alpha_i$, $e_i$ will not cross line $l$. Using a similar reasoning we can show that we can push $P$ down until it lies on the other side of $l$, while $e_i$ remains on the same side of $l$.

Finally if $e_i$ lies on $l$ we have $\phi < \pi/2$ and $\alpha_i = \pi - \phi$. First rotate $P$ into a vertical position, during which rotation $e_i$ does not move. We then rotate $P$ left or right; during this rotation we move $e_i$ until $e_i$ reaches the position of $e'_i$. We can maintain an angle of $\alpha_i$ between $e_{i-1}$ and $e_i$ since $\pi/2 < \alpha_i = \pi - \phi \leq \pi - \theta$ implies that the range $[\theta, \pi - \theta]$ contains $\alpha_i$. □

## 3 Chains with local angle restrictions

In this section, we show that if some simple relations between adjacent angles hold, a chain is flat-state connected. Define $\beta_i = \min(\alpha_i, \pi - \alpha_i)$.

**Theorem 2** *Chains with angles $\alpha_i$ such that $\alpha_i \leq \beta_{i-1} + \beta_{i+1}$ or $\alpha_i \geq \pi - \max(\beta_{i-1}, \beta_{i+1}) + \min(\beta_{i-1}, \beta_{i+1})$ for $2 \leq i \leq n-2$ are flat-state connected.*

**Proof.** (sketch) Let $C$ and $D$ be two flat configurations of a chain that satisfies the conditions of the lemma. Assume without loss of generality that $C$ is embedded in a plane $Q$. Let $P$ be a plane that is parallel to and lies above plane $Q$ at distance $\epsilon$.

At the start of iteration $i$ edges $e_0, e_1, \ldots, e_{i-1}$ lie in $P$. The edges $e_{i+1}, e_{i+2}, \ldots, e_{n-1}$ lie in $Q$. Edge $e_i$ connects $v_i$ in $P$ to $v_{i+1}$ in $Q$. The edges $e_0, e_1, \ldots, e_{i-1}$ conform to the layout of $D$, the edges $e_i, e_{i+1}, e_{i+2}, \ldots, e_{n-1}$ have the layout of $C$.

For all $i$ let $\gamma_i$ be the variable angle between $e_i$ and $P$ with $\gamma_i \leq \pi/2$. We move $P$ upward, $e_0, e_i, \ldots, e_{i-2}$ remain in $P$, $e_{i-1}$ drops below $P$, $e_i$ stays above $Q$ and $e_{i+1}, e_{i+2}, \ldots, e_{n-1}$ remain in $Q$. The motion continues, with angles $\gamma_{i-1}$ and $\gamma_i$ increasing, until $e_{i-1}$ and $e_i$ lie in a plane perpendicular to $P$. For an illustration, see Figure 3.

The motion of $e_{i-1}$ can then be reversed, while $e_i$ moves to its correct position. In other words, the layout before and after iteration $i$ can be moved to the same intermediate configuration, that in which

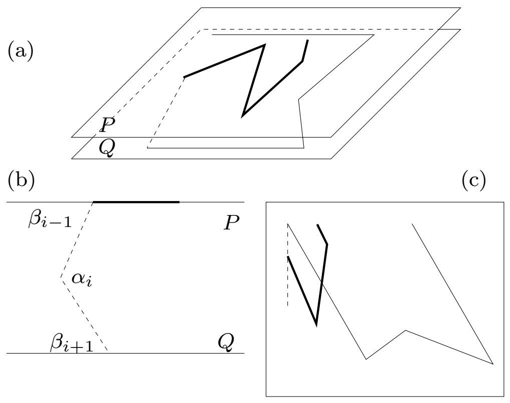


Figure 3: Edges in $P$ are bold; edges in $Q$ are solid; edges between $P$ and $Q$ are dashed. (a) planes $P$ and $Q$; (b) side view when $e_{i-1}$ and $e_i$ lie in plane perpendicular to $P$ and $Q$; (c) top view of the same configuration.

$e_{i-1}$ and $e_i$ lie in the perpendicular plane. This is sufficient to prove the theorem.

We first assume that $\beta_{i-1} \geq \beta_{i+1}$. If $\alpha_i \leq \beta_{i-1} + \beta_{i+1}$ let $g$ be such that $\alpha_i = g(\beta_{i-1}+\beta_{i+1})$. If $\alpha_i \geq \pi - (\beta_{i-1}-\beta_{i+1})$ let $g$ be such that $\alpha_i = \pi - g(\beta_{i-1}-\beta_{i+1})$. So $0 < g \leq 1$.

Suppose we have rotated the edge $e_{i-1}$ out of the plane $P$ such that $\gamma_{i-1} = f\beta_{i-1}$ with $0 < f \leq g$. Suppose we also have rotated $e_i$ with $\gamma_i = f\beta_{i+1}$. It is not hard to show that when $f = g$ we have reached the situation where $e_{i-1}$ and $e_i$ lie in a plane perpendicular to $P$. Also it can be shown that for any value of $f$ we can maintain the angles $\alpha_{i-1}$ and $\alpha_{i+1}$ by rotating the chains in $P$ and $Q$. What remains to be shown is that we can maintain the angle $\alpha_i$.

Below we show that $\alpha_i$ falls in the range $[f(\beta_{i-1} + \beta_{i+1}), \pi - f(\beta_{i-1} - \beta_{i+1})]$ for all values of $f$. This implies that we can maintain the angle $\alpha_i$ at $v_i$ by rotating $e_i$ and $Q$ around $v_i$ .

If $\alpha_i = g(\beta_{i-1} + \beta_{i+1})$, then $\alpha_i \geq f(\beta_{i-1} + \beta_{i+1})$. Since $g\beta_{i-1} \leq \pi - g\beta_{i-1}$ we have $\alpha_i \leq \pi - g\beta_{i-1} + g\beta_{i+1} \leq \pi - f(\beta_{i-1} - g\beta_{i+1})$. If $\alpha_i = \pi - (g\beta_{i-1} - \beta_{i+1})$, then $\alpha_i \geq g\beta_{i-1} + g\beta_{i+1} \geq f(\beta_{i-1} + \beta_{i+1})$. Also $\alpha_i \leq \pi - f(\beta_{i-1} - g\beta_{i+1})$.

To complete the proof we have to consider that case that $\beta_{i-1} < \beta_{i+1}$. However this case is similar to the case that $\beta_{i-1} \geq \beta_{i+1}$. So the theorem holds. □

The above theorem implies that any chain with $\delta \leq \alpha_i \leq 2\delta$ is flat state connected for any value of $\delta$ with $0 < \delta \leq \pi/3$.

## 4 Unit length chains

In this section we show that unit length chains with all dihedral angles in the range $(60°, 150°)$ are flat-state connected. Proofs for some of our claims are left out due to space constraints.

In a plane perpendicular to the original, we use a canonical configuration, defined as follows: the first edge $v_1v_2$ of the given chain must point up[2] . From there, each successive edge $v_iv_{i+1}$ is placed so that $v_{i+1}$ reaches a position with maximum height (without interfering with edges already fixed in place).

First we prove that a canonical chain is simple. Let the notation $v_a > v_b$ denote that $v_a$ is higher than $v_b$ in the vertical plane. We can show that in a canonical chain no edge can point down at a slope greater than 30° from horizontal, and two successive edges cannot both point down. Also, if $v_iv_{i+1}$ points up, then $v_{i+2}$ is at least half a unit higher than $v_i$. These two claims lead to the following two results:

**Lemma 3** *Once a particular height $h$ is reached by the canonical chain, the remaining chain cannot reach more than half a unit below $h$.*

**Corollary 4** *If $e_1$ and $e_2$ are consecutive edges that point up, no successor of $e_2$ can intersect $e_1$.*

**Lemma 5** *A canonical configuration has the property that every third vertex has monotonically increasing height. In addition, if edge $e_i$ points up, then $v_{i+3}$ is at least $\frac{1}{2}$ higher than $v_i$.*

**Proof.** Consider any three consecutive edges, $e_1$, $e_2$, $e_3$. We will show that $v_4$ must always be higher than $v_1$. If none of the three edges point down, then the claim holds trivially.

Suppose that $e_2$ points up. We know that a possible height decrease due to $e_1$ is less than $\frac{1}{2}$ , and that $e_2$ and $e_3$ combine to a height increase of at least $\frac{1}{2}$.

Instead, if $e_2$ points down, then the other two edges point up. Thus $e_1$ and $e_2$ increase height by at least $\frac{1}{2}$, and this increase cannot be negated by $e_3$.

If the first edge points up, then only one edge can point down so the latter can be combined with its predecessor for a net height increase of at least $\frac{1}{2}$. □

**Lemma 6** *Every six consecutive vertices result in a height increase of at least $\frac{1}{2}$.*

**Proof.** Let the six edges be $e_1, \ldots, e_6$. If $e_1$ or $e_4$ point up then by Lemma 5 there is a triplet of consecutive edges ($e_1e_2e_3$ or $e_4e_5e_6$) that gains $\frac{1}{2}$ in height. The other triplet does not lose height, so the claim is true. If both $e_1$ and $e_4$ point down, then $e_2$, $e_3$ and $e_5$ must point up . $e_6$ may point up or down. Thus pairs ($e_3e_4$) and ($e_5e_6$) each contribute a height increase of at least $\frac{1}{2}$. $e_2$ contributes positively, and $e_1$ can lose at most $\frac{1}{2}$. □

[2]We say that an edge $v_iv_{i+1}$ *points down* if $v_{i+1}$ is strictly lower than $v_i$. Otherwise the edge *points up*. Pointing left and right are defined similarly, with vertical edges symbolically defined to be pointing right.

Lemmas 3 and 6 imply that edge $e_{i+6}$ and its successors cannot intersect $e_i$ or its preceding edges. Thus what remains is to show that no six consecutive edges in canonical form can self-intersect.

From the angular restrictions of the problem definition, we know that no three consecutive edges can intersect. We can prove that no four or five consecutive edges in canonical form intersect. The proofs are similar to the following:

**Lemma 7** *Six consecutive edges in canonical form cannot intersect.*

**Proof.** We know that five consecutive edges do not intersect in canonical form. Thus we focus on proving that $e_1$ does not intersect $e_6$.

Case 1: $e_1$ points up. By corollary 4, $e_2$ must point down if there is to be an intersection. Thus $e_3$ points up. If $e_6$ points down (implying $e_5$ points up) we have $v_6 > v_7 > v_5 > v_2 > v_1$ so we are done. If instead $e_6$ points up, we must look at $e_5$: if it points up, we have $v_7 > v_6 > v_5 > v_2 > v_1$, implying no intersection. Thus the only dangerous configuration remaining has $e_5$ pointing down (and $e_4$ pointing up). We know that $v_6$ is at least $\frac{1}{2}$ higher than $v_4$, which is no more than $\frac{1}{2}$ lower than $v_2$. So $v_7 > v_6 > v_2 > v_1$.

Case 2: $e_1$ points down (so $e_2$ points up). By Lemma 5, $v_7 > v_4 > v_1$. Since $v_1 > v_2$, we must only prove that $v_6 > v_1$. Clearly we only worry if $e_6$ points up. Let us examine the pair $e_4e_5$. If $e_4$ points up, this would mean that $v_6 > v_4$, which proves our claim. So instead assume that $e_4$ points down, which means $e_3$ and $e_5$ point up. Since $v_5$ is at least $\frac{1}{2}$ higher than $v_3$, and $v_3$ is no more than $\frac{1}{2}$ lower than $v_1$, we have $v_6 > v_5 > v_1$. □

This concludes the proof of the following theorem:

**Theorem 8** *A chain that is in canonical form must be simple.*

**Theorem 9** *Any two planar chains with edges of unit length and angles in the range $(60°, 150°)$ are flat-state connected.*

**Proof.** Let the given chain be in the horizontal plane. We begin by lifting the first edge so that it projects vertically onto the second edge. Now suppose that we have part of the linkage still in the original configuration, and part of it has been lifted into a vertical plane and is in canonical form. We want to move the canonical portion of the chain into a position above the next edge of the horizontal portion, as demonstrated in Figure 4. On the left of the figure is a partially lifted chain. The result will be a configuration such as the one on the right. Two simultaneous dihedral motions are performed during this operation.

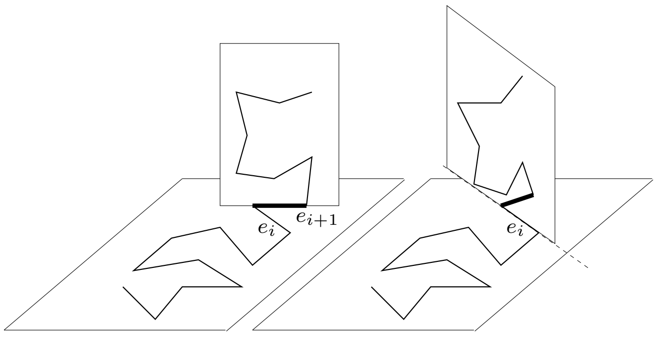


Figure 4: Lifting the next edge into canonical form.

Edges that are already in canonical configuration remain coplanar (in a vertical plane) throughout these motions. To do this, we rotate $e_{i+1}$ about $e_i$ and at the same time we rotate the canonical plane so that it always projects vertically through $e_i$. We call these two dihedral motions *primary*. We only need to intervene if an edge $u$ points directly *up* (becomes vertical) during the primary motion. At this instant the chain $C_u$ above $u$ may be placed arbitrarily in either of two possible positions in the canonical plane. If the overall motion is to continue, $u$ and the edge above it will no longer satisfy the greedy property. Thus we rotate $C_u$ about $u$ and proceed with the primary motion until it is complete or another edge becomes vertical. □

## References

[1] G. Aloupis, E. Demaine, V. Dujmovic, J. Erickson, S. Langerman, H. Meijer, J. O'Rourke, M. Overmars, M. Soss, I. Streinu, and G. Toussaint. Flat-state connectivity of linkages under dihedral motions. In *Proc. 13th Int. Symposium on Algorithms and Computation, volume 2518 of LNCS*, pp. 369–380, 2002.

[2] G. Aloupis, E. Demaine, H. Meijer, J. O'Rourke, I. Streinu, and G. Toussaint. On flat-state connectivity of chains with fixed acute angles. In *Proc. 14th Canadian Conference on Computational Geometry*, pp. 27–30, Aug. 2002.

[3] E. Demaine, S. Langerman, and J. O'Rourke. Geometric restrictions on producible polygonal protein chains. In *Proc. 14th Int. Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pp. 395–404, 2003.

[4] J. Erickson, M. Overmars, and M. Soss. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry: Theory and Applications*, 26(3):235–246, 2003.

[5] M. Soss. *Geometric and computational aspects of molecular reconfiguration.* PhD thesis, School of Computer Science; McGill University, 2001.

[6] M. Soss and G. Toussaint. Geometric and computational aspects of polymer reconfiguration. *Journal of Mathematical Chemistry*, 27(4):303–318, 2001.

# Gray Code Enumeration of Plane Straight-Line Graphs*

O. Aichholzer† F. Aurenhammer‡ C. Huemer§ B. Vogtenhuber¶

## Abstract

We develop Gray code enumeration schemes for geometric graphs in the plane. The considered graph classes include plane straight-line graphs, plane spanning trees, and connected plane straight-line graphs. Previous results were restricted to the case where the underlying vertex set is in convex position.

## 1 Introduction

Let $E = \{e_1, \ldots, e_m\}$ be an ordered set. For the purposes of this paper, $E$ will consist of the $m = \binom{n}{2}$ line segments spanned by a set $S$ of $n$ points in the plane, in lexicographical order. Consider a collection $\mathcal{A}$ of subsets of $E$. For instance, think of $\mathcal{A}$ being the class of all crossing-free spanning trees of $S$. We associate each member $A_i \in \mathcal{A}$ with its containment vector $b_i$ with respect to $E$. That is, $b_i$ is a binary string of length $m$ whose $j^{th}$ bit is 1 if $e_j \in A_i$ and 0, otherwise. A (combinatorial) *Gray code* for the class $\mathcal{A}$ is an ordering $A_1, \ldots, A_t$ of $\mathcal{A}$ such that $b_{i+1}$ differs from $b_i$ by a transposition, for $i = 1, \ldots, t-1$. For example (and as one of the results of this paper), for crossing-free spanning trees a Gray code exists such that successive trees differ by a single edge move. Depending on the class we will consider, a transposition will be an exchange of two different bits (as for the spanning tree class) or/and a change of a single bit. Combinatorial Gray codes generalize the classical binary reflected Gray code scheme [13] for listing $m$-bit binary numbers so that successive numbers differ in exactly one bit position. See [14] for a survey article. We also refer to [3] for various results concerning edge moves in spanning trees.

Any Gray code for a given class $\mathcal{A}$ provides a complete enumeration scheme for $\mathcal{A}$ by means of constant-size operations. Listing all the objects of a given class is a fundamental problem in combinatorics and, in particular, in computational geometry. Not every enumeration scheme constitutes a Gray code, however, as a small difference between consecutive objects will not be guaranteed, in general. For instance, the popular reverse search enumeration technique [6] lacks this property.

When interpreting $\{A_1, \ldots, A_t\}$ as the set of nodes of an abstract graph that connects two nodes $A_i$ and $A_j$ whenever $b_i$ and $b_j$ are a single transposition apart, any Gray code for the class $\mathcal{A}$ corresponds to a Hamiltonian path in this transposition graph. For example, if $\mathcal{A} = \mathcal{G}$, the class of all possible straight-line graphs on a point set $S$, then each of the $2^m$ subsets of $E$ defines a member of $\mathcal{G}$, for $m = \binom{n}{2}$. The transposition graph for $\mathcal{G}$ is the hypercube in $m$ dimensions.

While general enumeration schemes for geometric objects have been studied quite extensively, see e.g. [6, 2, 8, 7], respective results for Gray codes are sparse. In particular, all the known results for straight-line graphs concern the special case where the underlying set $S$ of points is in convex position [5, 9, 11, 10]. These Gray code constructions are mainly based on a hierarchy of graphs structured by increasing point set cardinality. In this paper, we extend this approach to general point sets $S$, which becomes possible when combining it with classical combinatorial Gray codes. We construct Gray codes for the class $\mathcal{PG}$ of all plane straight-line graphs, the class $\mathcal{CPG}$ of all plane and connected straight-line graphs, and the class $\mathcal{ST}$ of all plane spanning trees, for a given point set. For the class $\mathcal{CPG}$ no results existed even for the convex case. The respective challenging question for triangulations remains open (for $n \geq 7$).

## 2 A hierarchy for plane graphs

Let $S = \{p_1, \ldots, p_n\}$ be the underlying set of points in the plane. Without loss of generality, let $S$ be given in sorted order of $x$-coordinates. For simplicity, we also assume that no three points in $S$ are collinear. Then, for $1 \leq k \leq n-1$, the point $p_{k+1}$ lies outside the convex hull of $\{p_1, \ldots, p_k\}$. This property will turn out useful in the subsequent constructions.

Let now $\mathcal{A}$ be one of the classes $\mathcal{PG}$, $\mathcal{CPG}$, or $\mathcal{ST}$. We define a hierarchy (a tree structure) $H_{\mathcal{A}}(S)$ for $\mathcal{A}$ and $S$ such that the $k^{th}$ level of the hierarchy consists of all the members of $\mathcal{A}$ on top of the first $k$ points in $S$, for $k = 1, \ldots, n$. That is, each member in $H_{\mathcal{A}}(S)$ except the root (at level 1) has a unique par-

*Research supported by the FWF Joint Research Project Industrial Geometry S9205-N12 and Projects MCYT-FEDER BFM2003-00368 and Gen. Cat 2005SGR00692

†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 Matematica Aplicada II, Universitat Politecnica de Catalunya, Barcelona, Spain, `Huemer.Clemens@upc.edu`

¶Institute for Software Technology, University of Technology, Graz, Austria, `lambda@fsmat.at`

ent, and each member in $H_{\mathcal{A}}(S)$ which is not a leaf has a unique set of children.

The Gray code construction for $\mathcal{A}$ is done recursively. It hinges on an appropriate rule for defining the parent of a given member, as well as on a consistent rule for enumerating its children. Designing the enumeration rule for the children is the crucial part, as it has to yield a Gray code for the children which fits the (previously constructed) Gray code for the parents. In particular, $H_{\mathcal{A}}(S)$ has to be an ordered tree such that the ordering at each of its levels is a Gray code.

## 3 The class $\mathcal{PG}$

For the class $\mathcal{PG}$ of all plane straight-line graphs, things are surprisingly simple. Let $G'$ be at level $k+1$ of the hierarchy $H_{\mathcal{PG}}(S)$, for $k \geq 1$. That is, $G'$ is some plane straight-line graph whose vertex set is $\{p_1, \ldots, p_{k+1}\}$.

The parent of $G'$ is obtained by removing from $G'$ the vertex $p_{k+1}$ and all its incident edges. This gives a unique graph $G$ at level $k$ of $H_{\mathcal{PG}}(S)$. We say that a vertex $p_j$ of $G$ is *visible* (from $p_{k+1}$) if the line segment $p_{k+1}p_j$ does not cross any edge of $G$. As we are interested only in plane graphs, all the children of $G$ are obtained by adding the vertex $p_{k+1}$ and connecting $p_{k+1}$ to subsets of visible points in all possible ways (including the empty set).

Let $v_G$ be the number of visible vertices of $G$. We can use the cyclic binary $v_G$-bit Gray code $B(v_G)$ (see Appendix) for encoding all the possible subsets of edges incident to $p_{k+1}$. A transposition then corresponds to adding or removing a single edge to a visible vertex. To specify the first and the last subset (i.e., child of $G$), we take $\emptyset$ for the first child and the singleton set $\{p_{k+1}p_j\}$ for the last child, where $p_{k+1}p_j$ is the edge tangent to the convex hull of $G$ from above. (Vertex $p_j$ is visible for any choice of the graph $G$. In particular, $p_j$ is the first visible vertex in counterclockwise order.) Accordingly, the first string in the code $B(v_G)$ is $00\ldots0$ and the last string is $10\ldots0$.

To construct a Gray code for level $k+1$ of $H_{\mathcal{PG}}(S)$, let $G$ and $F$ be adjacent at level $k$ of $H_{\mathcal{PG}}(S)$. Attaching $00\ldots0$ to the string for $G$ and $F$, respectively, leaves the strings one transposition apart. The same is true for $10\ldots0$. That is, the first child of $G$ is adjacent to the first child of $F$, and the last child of $G$ is adjacent to the last child of $F$. So we can run $B(v_G)$ followed by $\overline{B(v_F)}$ (the code $B(v_F)$ in reverse order), and so on. The construction of the desired Gray code is now obvious. In addition, we can use the fact that the number of plane straight-line graphs on $n \geq 2$ points is even (because each convex hull edge appears in exactly half of the graphs), which by induction implies that the Gray code is cyclic. In the language of graph theory, our result reads:

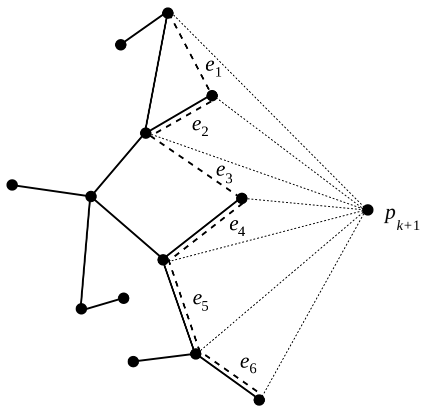


Figure 1: The chain of a spanning tree

**Theorem 1** *The transposition graph for $\mathcal{PG}$ contains a Hamiltonian cycle.*

It would be interesting to know the average degree of $p_{k+1}$ in a level-$(k+1)$ member of $H_{\mathcal{PG}}(S)$. This information could be used to give a lower bound on the number of plane straight-line graphs.

## 4 Plane spanning trees

Our strategy for constructing Gray codes also works for the class $\mathcal{ST}$ of plane spanning trees. A transposition will now be an exchange of two different bits, because the addition (or removal) of a single edge destroys the tree property.

Consider a member $T'$ at level $k+1$ of the hierarchy $H_{\mathcal{ST}}(S)$, for $k \geq 1$. That is, $T'$ is some plane spanning tree on $\{p_1, \ldots, p_{k+1}\}$. Defining an appropriate parent of $T'$ is less trivial now.

For an arbitrary plane straight-line graph $G$ on $\{p_1, \ldots, p_k\}$, let $q_1, \ldots, q_v$ be the visible vertices of $G$ (as seen from $p_{k+1}$) in counter-clockwise order. We define the *chain for* $G$, $C(G)$, as the ordered set of line segments $q_iq_{i+1}$, for $1 \leq i \leq v-1$. Observe that for every edge $e \in C(G) \setminus G$, the set of visible vertices of $G \cup \{e\}$ is the same as for $G$.

*Parent rule:* Let $G$ be the graph obtained by removing from $T'$ the vertex $p_{k+1}$ and all its incident edges. Let $G$ consist of $r \geq 1$ components. We add the first $r-1$ edges of the chain $C(G)$ that connect these components, and define the resulting tree as the parent of $T'$.

This rule yields a unique parent for $T'$. Notice that this parent is well-defined (i.e., belongs to level $k$ of $H_{\mathcal{ST}}(S)$) because $G$ can always be connected to a tree using edges of $C(G)$, and no such edge crosses any edge of $G$. From the definition of the parent we get the definition for the children, as follows.

Let $T$ be a tree at level $k$ of the hierarchy $H_{\mathcal{ST}}(S)$, and let '$<$' denote the total order on $C(T)$. Define $E(T)$ as the set of all edges $e \in T \cap C(T)$ which satisfy the following two conditions:

(1) Removal of $e$ does not make a non-visible vertex of $T$ visible.

(2) No edge $e' \in C(T) \setminus T$ with $e' < e$ gives a cycle in $T \cup \{e'\}$ that contains $e$.

See Figure 1. The spanning tree $T$ is drawn with bold lines. Its chain $C(T)$ consists of six edges $e_1, \dots, e_6$. Edges $e_1$ and $e_3$ are not part of $T$ and thus do not belong to $E(T)$. Also, we have $e_4, e_5 \notin E(T)$ because these edges reveal visible points. Finally, $e_2 \notin E(T)$ as the edge $e_1 < e_2$ closes a cycle in $T$ that contains $e_2$. This gives $E(T) = \{e_6\}$.

*Children rule:* The children of $T$ are obtained by removing, for each possible subset of $E(T)$, its edges from $T$ and connecting each resulting component to the vertex $p_{k+1}$ using a single edge to some visible vertex, in all possible ways.

It is clear that all the graphs constructed from $T$ in this way are plane spanning trees on $\{p_1, \dots, p_{k+1}\}$. To see that each member of level $k+1$ of the hierarchy $H_{\mathcal{ST}}(S)$ is generated exactly once by the children rule (provided all the members of level $k$ have so), we need the lemma below. Let us color the edges of $E(T)$ *green*, and the edges that connect to $p_{k+1}$ *red*.

**Lemma 2** *The parent rule and the children rule are consistent.*

**Proof.** Child → parent: Let $T'$ be some child constructed from $T$. Then $T'$ contains $r \geq 1$ red edges. If $r = 1$ then no green edge has been removed from $T$ for constructing $T'$. According to the parent rule, we now remove from $T'$ the vertex $p_{k+1}$, which correctly gives us the single component $T$. Now assume $r \geq 2$. Then $r-1$ green edges have been removed from $T$ to construct $T'$, leaving $r$ components. Let $K_i$ and $K_{i+1}$ be two of these components, such that there is a single (removed) green edge $e$ between them. There is no edge $e' \in C(T) \setminus T$ with $e' < e$ and which gives a cycle in $T \cup \{e'\}$ containing $e$. Thus, when removing from $T'$ the vertex $p_{k+1}$ and joining components of the resulting graph $G$ according to the parent rule, the edge $e$ is indeed the first edge of $C(G)$ that connects $K_i$ and $K_{i+1}$. (Otherwise we would have chosen $e'$ for removal instead of $e$.) Thus we correctly get $T$ from $T'$ again.

Parent → child: Let $T$ be the parent constructed from $T'$. $T$ is obtained by removing $p_{k+1}$ and joining the $r$ components of the resulting graph $G$ with edges of $C(G)$. Let $e$ be such an edge. Then $e$ is the first edge of $C(G)$ that connects the respective two components. Therefore, no edge $e' \in C(T) \setminus T$ with $e' < e$ gives a cycle in $T \cup \{e'\}$ that contains $e$. Moreover, because $e \in C(G) \setminus G$, we know that $G \cup \{e\}$ and $G$ have the same set of visible vertices. In conclusion, $e$ is indeed a green edge. Thus, by the children rule, $T'$ will be constructed as one of the children of $T$. $\square$

**Theorem 3** *The transposition graph for $\mathcal{ST}$ contains a Hamiltonian path.*

**Proof.** Let $T$ be a tree at level $k$. Define as the first child $T^f$ (respectively, as the last child $T^\ell$) of $T$ the tree obtained when adding to $T$ the upper (respectively, lower) tangent from $p_{k+1}$ to the convex hull of $T$. Note that $T^f$ and $T^\ell$ are well-defined children of $T$. We claim (and prove below) that the transposition graph for $\mathcal{ST}$ contains a path from $T^f$ to $T^\ell$ that contains all the children of $T$. The theorem follows because, for two neighboring trees $T$ and $U$ at level $k$, their first children $T^f$ and $U^f$, as well as their last children $T^\ell$ and $U^\ell$, are a single transposition apart.

To encode all the children of $T$, we have to consider each subset $Y \subset E(T)$ of green edges and, for fixed $Y$, all allowed distributions of red edges. Let $g = |E(T)|$. Similar to Section 3, the binary reflected $g$-bit Gray code $B(g)$ (let us call it the *green* code) is used for encoding all possible subsets of $E(T)$. Now consider a fixed subset $Y$. Let $T \setminus Y$ have $r \geq 1$ components. Given an arbitrary visible vertex $q_j$ in each component $K_j$, $1 \leq j \leq r$, there exists a Gray code $R(r)$ (the *red* code) that starts with the positions of the edges $q_1p_{k+1}, \dots, q_rp_{k+1}$ and that encodes all allowed positions of the red edges; see the Appendix.

Between every two transpositions in the green code we work off the red code. Care has to be taken when switching between the codes. If a green edge is added (i.e., two components are joined) then some red edge has to be removed. All other red edges stay at their positions, which are the starting positions for the subsequent red code. Similarly, if a green edge is removed (i.e., a component is split into two) then some red edge has to be added. Again, all other red edges stay at their positions which are the starting positions for the next red code.

The green code is cyclic, and thus we obtain a cyclic Gray code for the children of $T$ when combining the green and the red code. If now $T^f$ and $T^\ell$ are not adjacent in this code, this property can be enforced as follows: The green code is started at the subset $Y = \emptyset$ of $E(T)$ (recall that $T^f$ and $T^\ell$ arise as children for this subset) and the red code for $\emptyset$ is started with some red edge which is not a convex hull tangent. Such an edge exists because $T^f$ and $T^\ell$ are already adjacent, otherwise. We first do one transposition in the green code and, after having returned to $\emptyset$ with the combined code in the end, we do the red code for $\emptyset$. Cutting the obtained code, in which $T^f$ and $T^\ell$ are adjacent now, between these two trees gives the desired result. $\square$

If the Hamiltonian path in Theorem 3 both starts and ends with a first child (or with a last child), then we get a Hamiltonian cycle. Constructing such a cycle in the general case is left to further research.

The Gray code construction above can be modified to yield a Gray code for the class $\mathcal{CPG}$ of all connected plane straight-line graphs. Allowed transpositions then are an exchange of two different bits or a change of a single bit. Disallowing either type makes the transposition graph for $\mathcal{CPG}$ non-Hamiltonian. Details are given in a full version of this paper.

## 5 Algorithmic issues

To perform a Gray code enumeration of a given graph class $\mathcal{A}$, the hierarchy $H_{\mathcal{A}}(S)$ is traversed in preorder and its leaves are reported. For $\mathcal{A} \in \{\mathcal{PG}, \mathcal{ST}, \mathcal{CPG}\}$, each non-leaf member of $H_{\mathcal{A}}(S)$ has at least two children. Consequently, the time complexity is dominated by computing the leaves. When computing the children of a given parent $G$, the main tasks are calculating the chain $C(G)$ and the set $E(G)$ of green edges. Both tasks can be accomplished in $O(|S|)$ time, by traversing the faces that $G$ defines within its convex hull. We omit the details due to lack of space.

**Theorem 4** *Let $\mathcal{A}$ be any of the classes $\mathcal{PG}$, $\mathcal{ST}$, or $\mathcal{CPG}$, for a given set $S$ of $n$ points in the plane. A Gray code enumeration of $\mathcal{A}$ can be performed in $O(n)$ time per member.*

For the class $\mathcal{ST}$ this result is superior to the $O(n^3)$ result for (non-Gray code) enumeration in [6].

## 6 Discussion

For several classes of plane straight-line graphs their transposition graph fails to be Hamiltonian, in general. E.g., for the class of all triangulations of a point set $S$, with edge flips [12] as transpositions, there is a counterexample for $|S| = 6$. However, it still may be that Hamiltonicity is attained when $S$ is sufficently large. For pseudo-triangulations, see e.g. [1], the situation is unclear as well. Our conjecture is that the flip graph is Hamiltonian when both edge-exchanging *and* edge-inserting flips are admitted. The flip graph of minimum (or pointed) pseudo-triangulations and edge-exchanging flips is known to be Hamiltonian for all sets of up to 5 points. Our hierarchical approach possibly may be used to settle these problems.

With small adaptions, our enumeration scheme also works for point sets containing collinear points. Thus we can provide Gray codes for the respective graph classes on the grid.

The efficient generation of random spanning trees, plane graphs, or connected plane graphs is still an open problem. Progress can possibly be made by selecting random children for members in the respective classes.

## 7 Appendix

**Binary reflected Gray code.** The binary reflected Gray code $B(n)$ for $n$-bit numbers is defined recursively as follows. $B(1)$ is a list of two one-bit strings, namely $0, 1$. For $n \geq 2$, $B(n)$ is formed by taking the list for $B(n-1)$, prepending the bit 0 to every string, and then taking the list for $B(n-1)$ in reversed order and prepending the bit 1 to every string. Thus

$$B(n) = \begin{cases} 0, 1 & \text{if } n = 1 \\ 0 \cdot B(n-1) \ \circ \ 1 \cdot \overline{B(n-1)} & \text{if } n \geq 2 \end{cases}$$

with $\cdot$ and $\circ$ denoting character and list concatenation, respectively.

**Red Gray code.** For a given set $Y$ of green edges, let the graph $T \setminus Y$ consist of the components $K_1, \ldots, K_r$. Let component $K_j$ have $v_j$ visible vertices, and let $1, \ldots, v_j$ be any fixed ordering for the positions of these vertices. The red code $R(1)$ for the first component $K_1$ is given by $1, 2, \ldots, v_1$. Assume we are given a red code $R(s)$ for $K_1, \ldots, K_s$, for $s < r$. Then the red code $R(s+1)$ is given by

$$1 \cdot R(s) \ \circ \ 2 \cdot \overline{R(s)} \ \circ \ 3 \cdot R(s) \circ \cdots \circ v_{s+1} \cdot \overline{R(s)}$$

where the last sublist is $v_{s+1} \cdot R(s)$ if $v_{s+1}$ is odd.

## References

[1] O. Aichholzer, F. Aurenhammer, P. Braß, H. Krasser. *Pseudo-triangulations from surfaces and a novel type of edge flip.* SIAM Journal on Computing 32 (2003), 1621-1653.

[2] O. Aichholzer, F. Aurenhammer, H. Krasser. *Enumerating order types for small point sets with applications.* Order 19 (2002), 265-281.

[3] O. Aichholzer, F. Aurenhammer, F. Hurtado. *Sequences of spanning trees and a fixed tree theorem.* Computational Geometry: Theory and Applications 21 (2002), 3-20.

[4] M. Ajtai, V. Chvátal, M.M. Newborn, E. Szemerédi. *Crossing-free subgraphs.* Annals of Discrete Mathematics 12 (1982), 9-12.

[5] R. Arenas, J. Gonzalez, A. Marquez, M. Puertas Gonzalez. *Grafo de Grafos Planos de un Poligono Convexo.* Jornadas de Matematica Discreta y Algoritmica 4 (2004), 31-38.

[6] D. Avis, K. Fukuda, *Reverse search for enumeration*, Discrete Applied Mathematics 65 (1996), 21-46.

[7] S. Bereg. *Enumerating pseudo-triangulations in the plane.* Computational Geometry: Theory and Applications 30 (2005), 207-222.

[8] S. Felsner. *On the number of arrangements of pseudolines*, Discrete & Computational Geometry 18 (1997), 257–267.

[9] M. C. Hernando, F. Hurtado, A. Marquez, M. Mora, M. Noy. *Geometric tree graphs of points in convex position.* Discrete Applied Mathematics 93 (1999), 51-66.

[10] C. Huemer, F. Hurtado, M. Noy, E. Omana-Pulido. *Gray codes for non-crossing partitions and dissections of a convex polygon.* In: Proc. X Encuentros de Geometria Computacional, Sevilla, 2003.

[11] F. Hurtado, M. Noy. *Graph of triangulations of a convex polygon and tree of triangulations.* Computational Geometry: Theory and Applications 13 (1999), 179-188.

[12] F. Hurtado, M. Noy, J. Urrutia. *Flipping edges in triangulations.* Discrete & Computational Geometry 22 (1999), 333-346.

[13] F. Ruskey. *Simple combinatorial Gray codes constructed by reversing sublists.* Springer Lecture Notes in Computer Science 762 (1993), 201-208.

[14] C. Savage. *A survey of combinatorial Gray codes.* SIAM Review 39 (1997), 605-629.

# The Rotation Graph of k-ary Trees is Hamiltonian [*]

Clemens Huemer[‡] Ferran Hurtado[‡] Julian Pfeifle[‡]

**Abstract**

In this paper we show that the graph of $k$-ary trees, connected by rotations, contains a Hamilton cycle. Our proof is constructive and thus provides a cyclic Gray code for $k$-ary trees. Furthermore, we identify a basic building block of this graph as the 1-skeleton of the polytopal complex dual to the lower faces of a certain cyclic polytope.

## 1 Introduction

A $k$-ary tree is a rooted plane tree where each vertex has either $k$ children or no children. Let $R(m,k)$ denote the graph whose vertices are all $k$-ary trees with $m$ internal nodes, and where two trees are adjacent if they differ by a rotation, defined below. For $k=2$, Lucas [6] and later Hurtado and Noy [2] showed that the rotation graph of binary trees is Hamiltonian. There exists a well known isomorphism [11] between the flip graph of triangulations of a convex polygon [2] and the rotation graph of binary trees. This isomorphism generalises to an isomorphism between graphs whose node sets are dissections of a convex polygon into $m$ $k$-gons and $(k-1)$-ary trees with $m$ internal nodes, respectively. See Figures 1, 2 and 3. In fact, a rotation for $k$-ary trees can be defined using this isomorphism to dissections of a convex polygon. A more direct definition of a rotation given by Sagan [9] is recalled in the next section. Another definition of rotations from Korsh [3] does not carry over to the flip graph of dissections, meaning that rotations for $k$-ary trees do not always correspond to edge-flips in the graph of dissections.

As our main results, we present a cyclic Gray code [10] for $R(m,k)$ and identify the graph $C(m-1,k)$ of weak compositions of $m-1$ into $k$ parts as a basic building block of $R(m,k)$. Moreover, $C(m-1,k)$ turns out to be the 1-skeleton of the polytopal complex dual to the lower faces of a certain cyclic polytope.

There exist several Gray codes for $k$-ary trees. It is common to encode trees as a sequence of numbers, and then give a Gray code for these sequences; see [4, 1, 13]. Trees having a similar representation by numbers can have a very different natural structure. Our Gray code guarantees that adjacent trees in the list are very similar. To our knowledge no Gray code for $k$-ary trees based on rotations was known so far. Sagan [9] already proved that the flip-graph of rotations for $k$-ary trees is connected. We state our main result which is a direct consequence of Theorem 9.

**Theorem 1** *There exists a cyclic Gray code for $k$-ary trees with $m$ internal nodes in which consecutive trees differ by a rotation.*

## 2 Rotations for $k$-ary trees and dissections of a convex polygon

Let $D(m,k)$ be the graph of dissections of a convex $n$-gon into $m$ convex $k$-gons, for $k \geq 3$ and $m \geq 1$. For $D(m,k)$ to exist, it is necessary and sufficient that $n = m(k-2)+2$. Two dissections in $D(m,k)$ are connected by an arc if they differ in the placement of exactly one diagonal (a 'flip'). By [7], the number of vertices of $D(m,k)$ is $\frac{1}{m}\binom{(k-1)m}{m-1}$.

The flip graph of dissections $D(m,k)$ has already arisen in [12] in the guise of the dual graph of the simplicial complex $\Delta(m,k)$ of dissections of a polygon into $m$ convex $k$-gons, whose facets are the dissections and the vertices the diagonals. Tzanaki [12] proves that $\Delta(m,k)$ is vertex-decomposable, therefore shellable, and in consequence homotopy equivalent to a wedge of spheres; in fact, to a wedge of $\frac{1}{m}\binom{m(k-2)}{m-1}$ spheres of dimension $m-2$. In particular, $\Delta(m,3)$ is the (polar of the) associahedron [5], but for $k \geq 4$ and $m \geq 2$ the complex $\Delta(m,k)$ is not even isomorphic to the boundary complex of a PL sphere, much less a polytope.

We recall the definition of a rotation (i.e., an edge of the graph $R(m,k)$) given by Sagan [9]. A subtree $T_v$ of a plane $k$-ary tree $T$ *generated* by a vertex $v$ consists of $v$ and all its descendants. If $v$ is an internal vertex then we let $v_1, v_2, \ldots, v_k$ be its children listed left to right and let $T_{v_1}, T_{v_2}, \ldots, T_{v_k}$ denote the trees they generate, respectively. We focus on the child $x = v_i$ and consider the list $L(T_v, x)$ of pairwise disjoint subtrees

$$L(T_v,x) := T_{v_1}, T_{v_2}, \ldots, T_{v_{i-1}}, T_{x_1}, T_{x_2}, \ldots, T_{x_k},$$

$$T_{v_{i+1}}, T_{v_{i+2}}, \ldots, T_{v_k}$$

[*]Research partially supported by Projects MCYT BFM2003-00368, Gen.Cat. 2005SGR00692 and Acción Integrada España Austria HU2002-0010

[‡]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya. huemer.clemens@upc.edu, ferran.hurtado@upc.edu, julian.pfeifle@upc.edu

listed left to right in the order in which they are encountered in T (i.e., in depth-first order). Then a tree $\overline{T}$ is a flip (i.e., a rotation) of $T$, if it is isomorphic to $T$ outside of $T_v$ and there is some child $y$ of $v$ such that $L(T_v, x)$ and $L(\overline{T_v}, y)$ are isomorphic as ordered lists of rooted plane $k$-ary trees.

The graphs $D(m, k+1)$ and $R(m, k)$ are isomorphic. Figures 2 and 3 illustrate the isomorphism between $D(3, 4)$ and $R(3, 3)$.

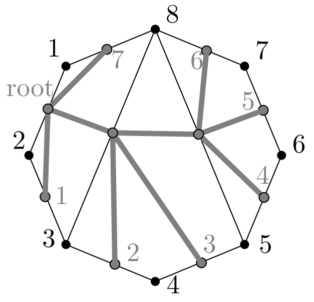


Figure 1: Bijection between a dissection of a convex polygon and a ternary tree.

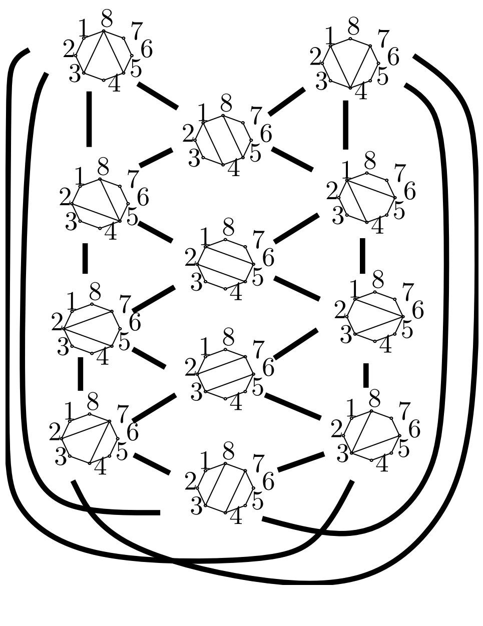


Figure 2: The graph $D(3, 4)$ is isomorphic to the graph $R(3, 3)$ of Figure 3.

## 3 Compositions

Let $r, s \geq 1$ be integers. A *(weak) composition of $r$ into $s$ parts* is an ordered $s$-tuple $(a_1, a_2, \ldots, a_s)$ of non-negative integers such that $a_1 + a_2 + \cdots + a_s = r$. We make the set $C(r, s)$ of all compositions of $r$ into $s$ parts into a graph by declaring two of them to be adjacent if they differ by one in exactly two positions that are connected by a (perhaps empty) sequence

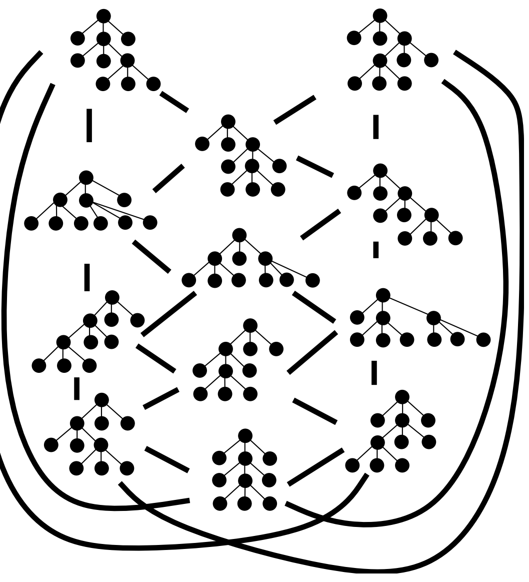


Figure 3: The graph $R(3, 3)$ is isomorphic to the graph $D(3, 4)$ of Figure 2.

of 0's. For example, the composition $(1, 0, 2, 4, 0, 1)$ is adjacent to $(1, 0, 2, 3, 0, 2)$, but not to $(0, 0, 2, 4, 0, 2)$.

**Proposition 2** *The graph $C(r, s)$ is isomorphic to the dual graph of the simplicial complex of lower faces of the $d$-dimensional cyclic polytope $C_d(r+d)$ on $r + d$ vertices, where $d = 2s - 2$.*

**Proof.** The two sets have the same cardinality, because by Gale's Evenness Criterion (see, e.g., [14]) the number of lower facets of $C_d(r+d)$ is $\binom{r+d-d/2}{d/2} = \binom{r+s-1}{s-1} = \#C(r, s)$. Next, we define an injective map from $C(r, s)$ to the set of (indices of vertices contained in) lower facets of $C_d(r+d)$. To a weak composition $r = a_1 + \cdots + a_s$, associate the subset of $\{1, 2, \ldots, r+d\}$ of size $d$ consisting of $s - 1$ pairs of consecutive integers surrounded by $s$ "holes", where the $i$-th "hole" has size $a_i$. For example, if $r = 3$ and $s = 4$, the composition $3 = 0 + 2 + 1 + 0$ corresponds to the set $\{1, 2, 5, 6, 8, 9\}$, which has "holes" $\emptyset$, $\{3, 4\}$, $\{7\}$, $\emptyset$, and indexes a lower facet of $C_6(9)$. This map is well-defined by Gale's Evenness Criterion, and injective by construction. Moreover, if two compositions are adjacent in $C(r, s)$, then exactly two "holes" in the corresponding facets of $C_d(r+d)$ differ in size by exactly $\pm 1$, respectively $\mp 1$. The two facets therefore share $d - 1$ vertices, and are thus adjacent. $\square$

**Remark 3** *The faces of dimension at most $s - 1$ of the complex dual to the lower faces of $C_d(r+d)$ form a polyhedral subdivision of the $r$ times dilated $(s-1)$-dimensional standard simplex $r\Delta^{s-1}$, where $\Delta^{s-1}$ is the convex hull of the unit vectors in $\mathbf{R}^s$. The graph*

*$C(r,s)$ is the 1-skeleton of this polyhedral decomposition, and the vertices of $C(r,s)$ are exactly the integer points of $r\Delta^{s-1} \subset \mathbf{R}^s$. Their coordinates correspond to the compositions of $r$ into $s$ parts (cf. Figure 4).*

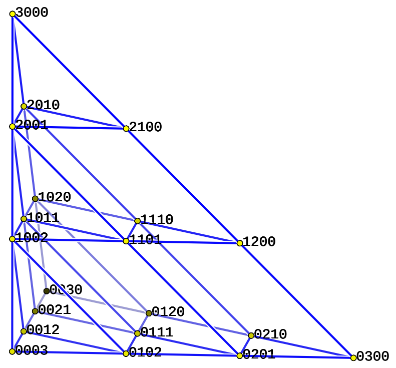


Figure 4: The graph $C(3,4)$ viewed as the 1-skeleton of a subdivision of a 3-dimensional simplex.

**Proposition 4** (see, e.g., [8]) *For each graph $C(r,s)$ with $r \geq s \geq 1$, there exists a Gray code $L(r,s)$ with endpoints $(r,0,\dots,0)$ and $(0,\dots,0,r)$.*

**Proof.** For $r \geq 1$, set $L(r,1) = r$. For $r \geq s > 1$, $L(r,s)$ is given by

$$L(r,s-1)\circ 0,\ \overline{L(r-1,s-1)}\circ 1,\ L(r-2,s-1)\circ 2,$$

$$\overline{L(r-3,s-1)}\circ 3, \dots,\ L(0,s-1)\circ r,$$

where $\overline{L(i,s-1)}$ is the list $L(i,s-1)$ in reverse order. $\square$

For example, the Gray code $L(3,3)$ for $C(3,3)$ is

$(300), (210), (120), (030), (021), (111), (201), (102), (012), (003)$.

## 4 A hierarchy for dissections

In [2] a hierarchy for triangulations was defined: all triangulations of convex polygons with any number of vertices are organized as nodes of a certain (infinite) tree. We generalise this approach and arrange all dissections of $n$-gons into $k$-gons into another rooted infinite tree $T_k$, which will be defined in the following by assigning a unique parent to every dissection but one. The vertices of each $D(m,k)$ are defined on polygons having $n = m(k-2)+2$ vertices, labelled in counterclockwise order. These vertices will lie on the level $m$ of $T_k$, so that the root of $T_k$, namely a $k$-gon, is at level 1. Thus, the dissections on level $m+1$ in this tree are defined on polygons having $n+k-2$ vertices $\{p_1,\dots,p_{n+k-2}\}$.

For $m \geq 1$ let $V = \{p_n, p_{n+1}, \dots, p_{n+k-2}\}$ be the set of "last" vertices of the polygons in $D(m+1,k)$. Fix a dissection $\delta_* \in D(m+1,k)$, and for all $p_{n+i} \in V$, let $a_i$ be the number of diagonals incident to $p_{n+i}$ in $\delta_*$. The total number of diagonals incident to vertices of $\delta_*$ in $V$ will be called $\ell = a_0 + \cdots + a_{k-2}$, so that the ordered tuple of non-negative integers $\mathbf{a}(\delta_*) = (a_0,\dots,a_{k-2}) \in C(\ell, k-1)$ is a composition of $\ell$ into $k-1$ parts.

**Definition 1** *The parent $\delta \in D(m,k)$ of a dissection $\delta_* \in D(m+1,k)$ is the dissection obtained by removing all $\ell$ diagonals in $\delta_*$ incident to vertices in $V$, placing $\ell$ new diagonals incident to $p_n$ into the "hole" created, and removing the $k-2$ vertices in $V \setminus \{p_n\}$. The* children *of a dissection $\delta \in D(m,k)$ are all dissections in $D(m+1,k)$ whose parent is $\delta$. A general child of $\delta$ will be denoted $\delta_*$. The* first child *$\delta_f$ of $\delta$ is the one associated to the composition $\mathbf{a}(\delta_f) = (\ell,0,\dots,0)$; the* last child *$\delta_l$ is associated to $\mathbf{a}(\delta_l) = (0,\dots,0,\ell)$.*

**Remark 5** *If there are $\ell-1$ diagonals incident to the last vertex $p_n$ of $\delta$, then the children of $\delta$ are obtained by adding the $k-2$ vertices in $V \setminus \{p_n\}$ to $\delta$, and distributing the new total of $\ell$ diagonals incident to $p_n$ among all vertices in $V$. In particular, the first child $\delta_f$ can be equivalently defined by placing a new $k$-gonal tile on top of the edge $p_1p_n$ and leaving all diagonals incident to $p_n$. Similarly, the last child $\delta_l$ is obtained from $\delta$ by placing a new tile on top of the edge $p_{n-1}p_n$ (thereby relabelling $p_n$ to $p_{n+k-2}$). Note that the reason that there are now $\ell$ diagonals incident to $p_n$ is that the former edge $p_1p_n$ of the convex hull is now a diagonal. Figure 5 shows all children of a dissection $\delta \in D(4,4)$.*

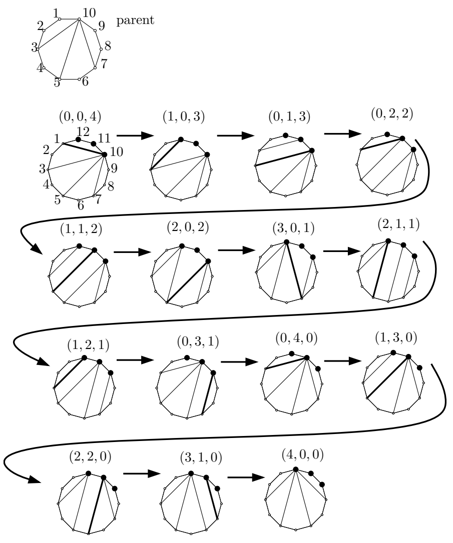


Figure 5: A Hamilton path for the children of a dissection according to a Gray code for compositions.

**Remark 6** *Because $D(m,k)$ inherits the symmetry of the $n$-gon it contains several copies of each compo-*

*sition graph $C(\ell, k-1)$, where $\ell$ is the total number of diagonals incident to some choice $V$ of "last" vertices of the $n$-gon. Moreover, because each $C(\ell, k-1)$ is isomorphic to the dual graph of the complex of lower facets of the cyclic polytope $C_{2k-4}(\ell + 2k - 4)$ by Proposition 2, the graph $C(\ell_1, k-1)$ is an induced subgraph of $C(\ell_2, k-1)$ whenever $\ell_1 \leq \ell_2$.*

**Lemma 7** *Let $\delta$ be any vertex of $D(m, k)$ and let $\ell - 1$ be the number of diagonals incident to the vertex $p_n$ of the dissection $\delta$. Then $C(\ell, k-1)$ is isomorphic to the subgraph of $D(m+1, k)$ induced by the children of $\delta$.*

**Proof.** We constructed the children of $\delta$ by distributing the diagonals which are incident to $p_n$ in $\delta$ among $V$. Every way of distributing the diagonals yields a unique child, and hence a unique composition. Thus, there is a bijection between the vertex sets. It is straightforward that an edge of the graph $C(\ell, k-1)$ corresponds to flipping one of the diagonals incident to a vertex of $V$, such that it remains incident to $V$. On the other hand, flipping such a diagonal away from $V$ yields a dissection $\delta_x$ which has a different parent. Since the number of diagonals incident to $V$ decreases, the composition assigned to $\delta_x$ does not belong to $C(\ell, k-1)$. Any two children of $\delta$ differ by edges incident to $V$, thus they are not adjacent by flipping an edge which is not incident to $V$. $\square$

## 5 A Hamilton cycle

In the following adjacency between two dissections is denoted with '$\sim$'.

**Lemma 8** *Let dissections $\delta$, $\delta^1$ and $\delta^2$ of $D(m, k)$ be given.*
*Then $\delta^1 \sim \delta^2$ implies $\delta^1_f \sim \delta^2_f$ and $\delta^1_l \sim \delta^2_l$. Moreover, there exists a Hamilton path formed by the children of $\delta$ whose endpoints are $\delta_f$ and $\delta_l$.*

**Proof.** The first statement follows immediately from Remark 5. The Hamilton path for the children of $\delta$ follows from Lemma 7 and Proposition 4. The endpoints of this path are the compositions $(0, \ldots, 0, \ell)$ and $(\ell, 0, \ldots, 0)$, where $\ell - 1$ is the number of diagonals incident to $p_n$ of $\delta$. These two compositions correspond to the dissections $\delta_f$ and $\delta_l$. $\square$

Figure 5 shows a Hamilton path among the children of a dissection and the corresponding Gray code for compositions. The diagonal which is exchanged in each step is drawn in bold.

**Theorem 9** *The graph $D(m, k)$ contains a Hamilton cycle for all $k \geq 3, m \geq 1$, with the exception of $D(2, 3)$ .*

The proof of Theorem 9 is based on Lemma 8. We omit the details in this abstract.

## 6 Acknowledgements

The authors thank Oswin Aichholzer for valuable discussions on the presented subject.

## References

[1] D.R. von Baronaigien, A Loopless Gray-Code Algorithm for Listing k-ary Trees, J. Algorithms 35 (2000) 100-107

[2] F. Hurtado, M. Noy, Graph of triangulations of a convex polygon and tree of triangulations, Computational Geometry: Theory and Applications 13 (1999) 179-188

[3] J. F. Korsh, Loopless generation of k-ary tree sequences, Information processing letters 52 (1994) 243-247

[4] J. F. Korsh, P. LaFollette, Loopless generation of Gray codes for k-ary trees, Information processing letters 70 (1999) 7-11

[5] C.W. Lee, The associahedron and triangulations of the $n$-gon, European J. Combinatorics 10 (1989) 551-560

[6] J.M. Lucas, The rotation graph of binary trees is Hamiltonian, J. Algorithms 8 (1987) 503-535

[7] J. H. Przytycki, A. S. Sikora, Polygon Dissections and Euler, Fuss, Kirkman, and Cayley Numbers, J. Combin. Theory, Ser. A, Volume 92 (2000) 68-76

[8] F. Ruskey, Simple combinatorial Gray codes constructed by reversing sublists, Lecture Notes in Computer Science, 762 (1993) 201-208.

[9] B. E. Sagan, Proper partitions of a polygon and k-Catalan numbers, preprint, `arXiv:math.CO/0407280`

[10] C. Savage, A survey of combinatorial Gray codes, SIAM Review 39 (1997) 605-629

[11] D. D. Sleator, R. E. Tarjan, W. P. Thurston, Rotation Distance, Triangulations, and Hyperbolic Geometry, J. American Mathematical Society 1 (3) (1988) 647-681

[12] E. Tzanaki, Polygon dissections and some generalizations of cluster complexes, preprint (2005), 9 pages, `math.CO/0501100v2`

[13] L. Xiang, K. Ushijima, C. Tang, On generating k-ary trees in computer representation, Information processing letters 77 (2001) 231-238

[14] G.M. Ziegler, Lectures on Polytopes, Graduate Texts in Mathematics, Vol. 152, Springer, Berlin 1995

# Cover Contact Graphs

M. Abellanas*, N. Atienza†, N. de Castro†, C. Cortés†,
M.A. Garrido†, C.I. Grima†, G. Hernández*, A. Márquez†, A. Moreno†,
J.R. Portillo†, P. Reyes†, J. Valenzuela† and M.T. Villar†

## Abstract

We study properties of the cover contact graphs (CCG). These graphs are defined by a pair $G = (S, C)$, where $S$ is a set of objects (called seeds) in the plane, and $C$ is a collection of discs or triangles (called covers) covering the seeds, with the property that the interior of those discs are mutually disjoint. The contact graph of that cover set is a CCG. Then we study three basic properties of CCGs taking into account the nature of the seeds and of the covers. Namely, whether there exists a connected CCG on a fixed seed set. Whether is it possible to realize a given graph as a CCG, and finally we try to enumerate certain classes of CCGs.

## 1 Introduction

In two disciplines very much related between them as Computational Geometry and Graph Drawing there exists a very broad literature on covering problems and intersection graphs. In the first case, we try to cover some objects with covers attending some properties. And in the second case, the vertex set of a graph is a collection of objects and the edges represent their intersections. Thus is not surprising that sometimes both problems appear at the same time and then we want to cover a given collection of objects and to consider the intersection graph of the covers. This is our case. A first example of this kind of problems appears in Koebe's Theorem (rediscovered many times. In this sense, check [10] for a very interesting report on Koebe's theorem and its interconnections) that establishes that any plane graph can be realized as a coin graph. When in a coin graph we cover a point set in the plane by discs centered at the points, the discs must have disjoint interiors, therefore the intersection is just a tangent contact. In this paper, we study the same class of graphs but we do not assume that the discs are centered at the points. Changing this condition, the properties of the graphs so obtained also change a lot as we will see later. Additionally, we cover other objects as discs and triangles by discs and triangles respectively and we see that again a very different behavior can be observed (some other similar representations have been considered previously, for instance in [3] not only vertices are represented by discs but also the faces).

More formally, as it is established in the abstract above, a cover contact graphs (CCG) is defined by a pair $G = (S, C)$, where $S$ is a set of objects (called seeds) in the plane, and $C$ is a collection of discs or isothetic triangles (called covers) covering the seeds, with the property that the interior of those discs (or triangles) are mutually disjoint. The contact graph of that cover set is a CCG. In other words, we join two seeds by an edge if their covers are tangent (or their borders have a non-empty intersection). An example of a CCG is depicted in Figure 1.

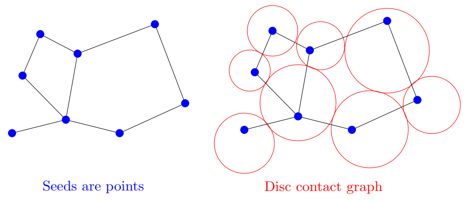


Figure 1: Realization of a graph

Not surprising, changing the nature of the seeds and of the covers different properties are obtained. We study here three of the more basic problems. Namely, *(1)* Whether there exists a connected CCG on a fixed seed set. *(2)* Whether is it possible to realize a given graph as a CCG. *(3)* We try to enumerate certain classes of CCGs. The first question is inspired by a result that says that given a point set on the plane to decide if there exists a connected coin graph on it, is an NP-complete problem [1]. In fact, even to decide if some coins can cover a given set is an NP-complete problem [2]. On the other hand, the second and the third problems are key problems from the point of view of Graph Drawing and Combinatorics respectively.

## 2 The seeds are points

As it is said in the Introduction, we make the first distinction attending the nature of the seeds, and, of course, the first class of seeds to be considered is that of points.

*U.P.M. (Spain)
†Universidad de Sevilla (Spain)

### 2.1 Points in the plane

In this subsection, the seeds will be always points in the plane and the covers will be either discs containing the points, or equilateral triangles with an edge parallel to the abscissae axis and the seed must be its bottom vertex.

Firstly, regarding connectivity, it is easy to see that, on the opposite of what happened for coin graphs, for any seed set it is always possible to find a connected CCG on it.

**Proposition 1** *For any seed set, there exists always a connected CCG on it (for both cases of covers, discs and triangles). Furthermore, it is possible to construct one of them in $O(n \log n)$ (being $n$ the number of points). Even more, for any seed set, there exists always 2–connected CCGs on it (for both cases of covers, discs and triangles). Furthermore, it is possible to construct one of them in $O(n^2)$ (being $n$ the number of points).*

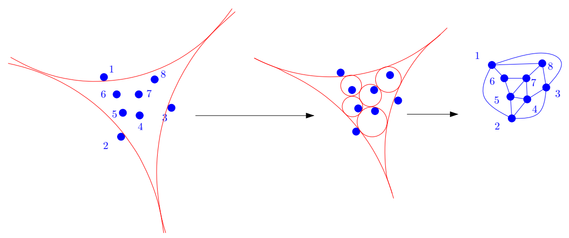


Figure 2: A 2-connected CCG

Focussing on realization, although we have seen that to find a connected CCG is always possible if the seeds are points in the plane, we will see now that to realize a given graph is much more difficult. Of course, if we do not fix the seeds, given a planar graph $G$, Koebe's theorem [9] guaranties that we can find a seed set $S$ such that it is possible to realize $G$ on $S$. But if the seeds are fixed previously, then the problem turns to be a very difficult one.

**Theorem 2** *Given a point set $S$ in the plane, and a planar graph $G$, to decide whether there exists a CCG on $S$ isomorphic to $G$ is an NP-complete problem.*

Nevertheless in some cases it is possible to say something more. The next result give some necessary conditions on a graph to be realizable. In order to establish that result we define a graph on a seed set $S$ inspired in the Sphere of influence graph defined by Toussaint [11] (see also [6, 7] for more results in the sphere of influence graphs). Given a point $p$ of a seed set $S$, we associate to $p$ the union $C(p)$ of all the maximal empty (in the sense that do not contain any seed in their interior) circles centered at vertices of its Voronoi region, the intersection graph of the sets so defined will be called the hyperinfluence graph of $S$ (denoted $HI(S)$).

**Proposition 3** *Let $G$ be a graph realizable as a CCG on a seed set $S$. Then (1) $G$ is a subgraph of $HI(S)$; (2) it is possible to give a plane representation of $G$ with $S$ as its vertex set and each edge with at most two rectilinear segments (one bend per edge).*

The first condition of Proposition 3, says us that graphs as that depicted in Figure 3 cannot be realizable as a CCG. And it is worthy to mention that the second condition will be used later to characterize CCGs when the seeds lie on a line.

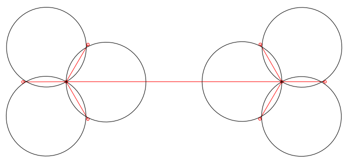


Figure 3: A non–realizable graph

To the light of Figure 3 one can ask what is the minimum number of seeds with a non-realizable graph. In this way we can enunciate

**Proposition 4** *Any graph with 4 or less vertices is representable as a CCG on any seed set.*

On the other hand, it is possible to give a collection of six points in convex position, such that its Delaunay triangulation is not representable as a CCG, see Figure 4.

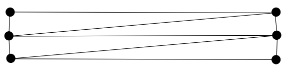


Figure 4:

Finally, regarding enumeration, as any planar graph can be realized by Koebe's theorem the remarkable result obtained in [4] holds in this case, namely, there exists $g \cdot n^{-7/2}\gamma^n n!$ realizable graphs of $n$ vertices, where $g \approx 0.4970043999 \cdot 10^{-5}$ and $\gamma \approx 27.2268777685$ are constants, given by explicit analytic expressions.

### 2.2 Points on a line

If we restrict the points to be on a line, then we can be more precise in the results obtaining a more accurate idea of what can be done and what cannot be done. In fact, we introduce an interesting particular case of CCGs. If we denote by $\Re^2_+$ to the half plane defined by the points with non-negative ordinate, we call a CCG$^+$ graph to a pair $G = (S, C)$, where $S$ is a set of objects (called seeds) in $\Re^2_+$ such that each seed has at least one point on the line $y = 0$, and $C$ is a collection of discs or isothetic triangles (called covers) in $\Re^2_+$ covering the seeds, with the property that the interior of those discs (or triangles) are mutually disjoint. The contact graph of that cover set is a CCG$^+$. (Figure 5) shows a CCG$^+$ graph.

We will follow the same structure as in the previous subsection.

To achieve the connectivity is an easy task when the points lie on a line as we see in the next result.

**Proposition 5** *Let $S$ be a set of $n$ seeds on a straight line, then* (1) *there exists always a realizable (as a CCG) $C_n$ (cycle of length $n$) on it;* (2) *there exists always a tree on $S$ realizable as CCG$^+$.*

About realization, we have seen in Proposition 5 that on any seed set, $C_n$ is always realizable. One can ask if Koebe's theorem is still valid when the seeds lie on a line, but this is not true since in [8] is proved that there is a plane triangulated graph with only 12 vertices such that for every placement of the vertices on a straight line at least one edge must bend at least twice in the resulting drawing.

This result together with Proposition 3 imply that the plane graph described in that paper is not realizable as a CCG if the seeds lie on a straight line. And, it is not difficult to see that any tree is realizable.

**Proposition 6** *Given any tree $T$, it is possible to choose seeds in a line such that $T$ is realizable as a CCG$^+$ on that seed set.*

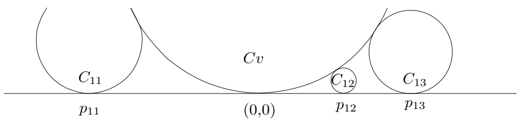


Figure 5: First step in realization of a tree $T$.

In Proposition 6 the seeds are not fixed on the line, an step beyond this situation is when the seeds are not fixed but their ordering on the line. In this case, not every tree is realizable as a CCG.

Given a labeled tree $T$, and an ordering $\mathcal{S}$ of the vertices of the tree, we say that $T$ is realizable on $\mathcal{S}$ if there exist points on a line such that $T$ realizable on those points and the ordering of the vertices of the tree on the line is $\mathcal{S}$.

A subgraph as that depicted in Figure 6 is called a forbidden chain. More precisely, given a graph $G = (V, E)$ an edge $e \in E$, a *forbidden chain for* $e$ is a matching $M = (V', E')$ of $G$ described as follows:

1. The vertex set $V'$ is a sequence of points in $\Re$, $A_1 < A_2 < B_1 < A_3 < B_2 < A_4 < \cdots < A_{2n-1} < B_{2n-2} < A_{2n} < B_{2n-1} < B_{2n}$

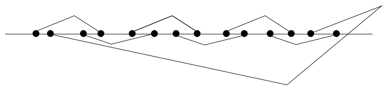


Figure 6: A forbidden chain.

2. For any $i \in \{1, 2, \ldots, 2n\}$, $\{A_i, B_i\} \in E'$.
3. $e = \{E_1, E_2\}$ has its end points separated by $M$ like one of these cases:
   - $e_1 \in (-\infty, A_2)$ and $e_2 \in (A_{2n}, B_{2n-1})$
   - $e_1 \in (-\infty, A_1)$ and $e_2 \in (A_2, B_1)$
   - $e_1 \in (A_2, B_1)$ and $e_2 \in (B_{2n-1}, +\infty)$
   - $e_1 \in (A_1, A_2)$ and $e_2 \in (B_1, B_2)$
   - $e_1 \in (A_2, B_1)$ and $e_2 \in (B_2, +\infty)$
   - $e_1 \in (-\infty, A_{2n-1})$ and $e_2 \in (A_{2n}, B_{2n-1})$

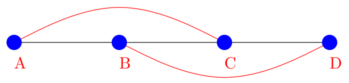


Figure 7: $T$ is not realizable on $S$. $\{A, C\}, \{B, D\}$ is an elemental forbidden chain

**Theorem 7** *Let $T$ be a labeled tree, and $\mathcal{S}$ and ordering of its vertices. Then, the following conditions are equivalent:* (1) *$T$ is realizable on $\mathcal{S}$ as a CCG;* (2) *it is possible to draw $T$ with its vertices on a line with only one bend per edge and such that the ordering of those vertices is $\mathcal{S}$;* (3) *there are no forbidden chains.*

As far as the third condition can be checked in linear time, we obtain

**Corollary 8** *Let $T$ be a labeled tree, and $\mathcal{S}$ and ordering of its vertices. Then, it can be decided in linear time whether $T$ is realizable on $\mathcal{S}$ as a CCG or not.*

A similar result can be establish regarding CCG$^+$.

**Theorem 9** *Let $T$ be a labeled tree, and $S$ and ordering of its vertices. Then, the following conditions are equivalent:* (1) *$T$ is realizable on $S$ as a CCG$^+$;* (2) *it is possible to draw $T$ in $\Re^2_+$ with its vertices on a line with only one bend per edge and such that the ordering of those vertices is $S$;* (3) *there are no elemental forbidden chains.*

**Corollary 10** *Let $T$ be a labeled tree, and $\mathcal{S}$ and ordering of its vertices. Then, it can be decided in linear time whether $T$ is realizable on $\mathcal{S}$ as a CCG$^+$ or not.*

So, once we have established results about realizability firstly without fixing the seeds, and then fixing the order in which they appear, we can give a result that describes how are all the CCG$^+$'s when the covers are triangles.

**Proposition 11** *Given a seed set $S$ on the line, and covers triangles isothetic to the triangle with vertices in the points $(0,0)$, $(-1,1)$, and $(1,1)$. Then, any $T$ realizable as a CCG$^+$ on it can be obtained following the next recursive method. Starting with $L$=$S$ and while $|L| > 1$, choose the closest pair $(u,v)$ of $L$ as an edge of $T$ and delete from $L$ either $u$ or $v$.*

Although the result of Proposition 11 is stated for a very particular case of triangles, it can be extended easily to any kind of triangles, changing the metric between the points.

The keys to enumeration are some results on realization. Namely, Propositions 6 and 11 and Theorem 9. Thus we can establish

**Theorem 12** *The following results hold for seeds on a line:* (1) *The number of labeled trees with $n$ vertices realizable as a CCG (or CCG$^+$) is $n^{n-2}$ (all the labeled trees);* (2) *given an ordering $\mathcal{S}$ of the natural numbers from 1 to $n$. The number of labeled trees realizable as a CCG$^+$ on $\mathcal{S}$ is the Catalan number $C_{n-1}$);* (3) *given a seed set $S$ on a line. The number of labeled trees realizable as a CCG$^+$ on $S$ is $2^{n-2}$.*

## 3 The seeds are discs or triangles

In this section, we consider either discs or isothetic triangles in the plane as a seed set, and we will cover them with the same kind of objects, this is to say, the covers for discs are discs and the covers for triangles are isothetics triangles.

Regarding connectivity, we have

**Proposition 13** *If the seeds are triangles with its bottom vertex in a horizontal line. Then there exists always a connected CCG$^+$ on it.*

We cannot extend the result of Proposition 13 any further. So in the case of triangles we can give collections of seeds such that no 2-connected CCG can be constructed on it (see Figure 8 (a)). And, in the case of discs, collections of seeds such that no connected CCG$^+$ can be constructed on it, as Figure 8 (b) shows. Even, it is not difficult to translate the example of Figure 8 (b) to CCGs in general; in order to get this goal it suffices to consider a huge disc just under all the other discs.

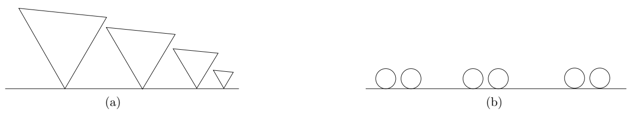


Figure 8: This seeds cannot be covered by a connected CCG$^+$.

Focussing on realization and enumeration, it is trivial to see that the result of Theorem 2 is still valid if we consider as discs as seeds instead of points (in order to see this remark it suffices to substitute the points by small discs). In fact, the results of Proposition 3 are also easily adaptable to the case of a set of discs as the seed set.

Regarding enumeration, the results obtained for points are clearly upperbounds in the case of discs or triangles, and the exact enumeration of the CCG representable on a concrete seed set seems to be a very difficult task.

## References

[1] M. Abellanas, N. de Castro, G. Hernández, A. Márquez and C. Moreno-Jiménez. *Gear System Graphs*, Preprint.

[2] M. Abellanas, S. Bereg, F. Hurtado, A. Garcia Olaverri, D. Rappaport and J. Tejel. *Moving Coins*, To appear in Comp. Geom. Theor. and Appl.

[3] G.R. Brightwell and E. R. Scheinerman. *Representations of Planar Graphs*, SIAM J. Disc. Math. Vol 6, No 2, (1993), pp 214-229.

[4] O. Giménez and M. Noy *The number of planar graphs and properties of random planar graphs*, Int. Conf. on Anal. of Alg. DMTCS proc. (2005), pp 147-156.

[5] M.R. Garey and D.S. Johnson. *Computing and Intractability, A guide to the theory of NP-completeness*, Freeman (1979).

[6] F. Harary and M.S. Jacobson and M.J. Lipman and F.R. McMorris. *Abstract Sphere-of-Influence Graphs*, Mathl. Comput. Modeling Vol. 17, No 11, (1993), pp 77-83.

[7] M.S. Jacobson and M.J. Lipman and F.R. McMorris. *Trees that are Sphere-of-Influence Graphs*, Appl. Math. Lett. 8, (1995), pp 89-93.

[8] M. Kaufmann and R. Wiese. *Embedding vertices at points: Few bends suffice for planar graphs*, J. of G. Alg. and Appli., vol 6, No 1, (2002), pp 115-129.

[9] Janos Pach and Pankaj K. Agarwal. *Combinatorial Geometry* John Wiley and Sons (1995).

[10] H.Sachs *Coin graphs, polyhedra and conformal mapping*, Disc. Math. 134, (1994) pp 133-138.

[11] G. Toussaint *Proximity graphs for nearest neighbor decision rules: recent progress*, Interface-2002, Canada, (2002).

# A binary labelling for plane Laman graphs and quadrangulations

Clemens Huemer[*] Sarah Kappes[†]

**Abstract**

We provide binary labellings for the angles of quadrangulations and plane Laman graphs which are in analogy with Schnyder labellings for triangulations [W. Schnyder, Proc. 1st ACM-SIAM Symposium on Discrete Algorithms, 1990].

## 1 Introduction

Schnyder-labellings are by now a classical tool to deal with planar graphs. One of their first applications was to obtain convex drawings of such graphs on a small grid [14].

A Schnyder-labelling is a special labelling of the angles of a plane graph with three colors. Schnyder [14] introduced this concept for triangulations, or maximal planar graphs. The angle-labelling corresponds directly to a decomposition of the edge-set into three spanning trees, or a *Schnyder-wood*. Felsner adapted this idea for 3-connected planar graphs [4].

Other classes of planar graphs, such as maximal bipartite planar graphs and *planar Laman graphs*, admit a decomposition of the edge set into two trees. Our motivation for this work was to obtain a binary labelling for these classes of graphs analogous to Schnyder's.

Let us recall that a graph is *planar* if it can be embedded in the plane; a *plane graph* has already been embedded in the plane [7]. We remark that for a binary labelling of a plane graph $G$, it is not necessary that $G$ is embedded with straight-line edges. We use only the combinatorics of the embedding, i.e. incidences of vertices, edges and faces, for our proofs.

A *quadrangulation* is a 2-connected plane graph where each interior face has four edges. A quadrangulation $Q$ is a maximal bipartite plane graph if and only if the outer face of $Q$ has four edges.

A tree decomposition for maximal bipartite planar graphs has been obtained by several authors [13, 12, 6, 1, 10]. The binary labelling "inherits"

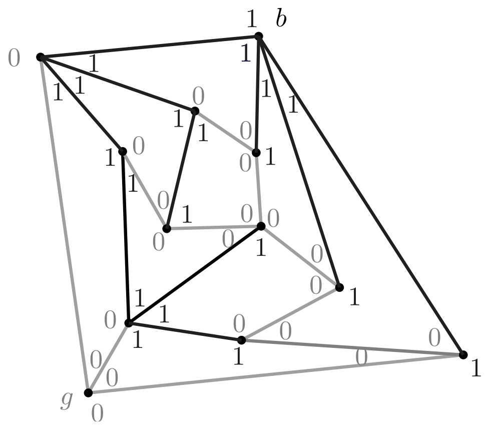


Figure 1: A binary labelling for a quadrangulation.

the tree decomposition property from Schnyder's labelling. More precisely, we obtain that every quadrangulation can be decomposed into two spanning trees by duplicating edges of the outer face.

Fraysseix and Mendez [5] relate Schnyder labellings for triangulations to 3-orientations and shelling orders. They also consider "separating decompositions" for maximal bipartite planar graphs which are closely related to binary labellings.

We show a binary labelling for plane Laman graphs. Laman graphs [9] are well known in the context of rigidity theory. A Laman graph on $n$ vertices contains $2n-3$ edges and every subgraph which is induced by $k$ vertices contains at most $2k-3$ edges.

Every planar Laman graph can be embedded as a *pointed pseudo-triangulation*[8]. A vertex $v$ of a plane straight-line graph is called *pointed* if it has an incident angle greater than $\pi$. A pointed pseudo-triangulation is a maximal pointed plane straight-line graph; this means that every vertex is pointed and adding any (non-crossing) edge yields a non-pointed vertex. In particular, the binary labelling holds for pointed pseudo-triangulations.

Other labellings for planar graphs have been investigated. Haas et al. [8], also see [11], defined "combinatorial pseudo-triangulations". Souvaine and Tóth [15] defined a vertex-face assignment for plane graphs.

Quadrangulations and plane Laman graphs are structurally different, because quadrangulations are bipartite graphs, whereas every pointed pseudo-

[*]Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya. huemer.clemens@upc.edu. Research supported by Projects MCYT BFM2003-00368 and 2005SGR00692.

[†]Department of Mathematics, Technical University Berlin, kappes@math.tu-berlin.de. This research was supported by the Deutsche Forschungsgemeinschaft within the European graduate program 'Combinatorics, Geometry, and Computation' (No. GRK 588/2).

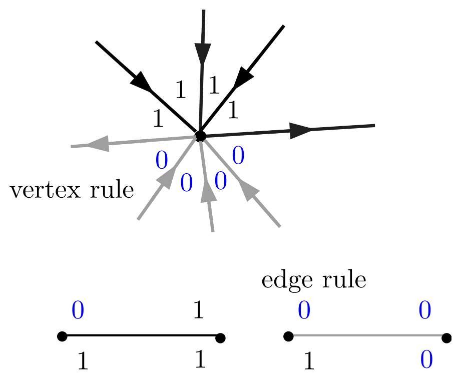


Figure 2: The vertex rule and the edge rule of a binary labelling.

triangulation contains a triangle. However, adding a (non-crossing) edge to a maximal bipartite plane graph yields a plane Laman graph, which in turn can be embedded as a pointed pseudo-triangulation. Thus, we believe that many concepts for pseudo-triangulations apply to quadrangulations and vice versa. The binary labelling only represents one aspect of this interesting fact. Another example is that every maximal bipartite planar graph can be obtained via a "Henneberg construction" [17] and also via "vertex splitting" [3]. As a last step of the construction one edge has to be deleted.

## 2 The binary labelling

Let a plane graph $G$ be given. A *binary labelling* for $G$ is a mapping from the angles of $G$ to the set $\{0, 1\}$ which satisfies the following conditions:

1. There are two special vertices $g$ and $b$ on the outer face of $G$. All angles incident to $g$ are labelled 0, all angles incident to $b$ are labelled 1.

2. **Vertex rule** For each vertex $v \notin \{b, g\}$, the incident labels form a non-empty interval of 1*s* and a non-empty interval of 0*s*.

3. **Face rule** For each face (including the outer face) its labels form a non-empty interval of 1*s* and a non-empty interval of 0*s*.

4. **Edge rule** For each edge its labels either are $0, 1 - 1, 1$ or $0, 1 - 0, 0$; see Figure 2.

**Observation 1** *The labelling of the edges induces an orientation: every edge is oriented towards its endpoint* $0, 0$, *respectively* $1, 1$. *In a binary labelling every vertex but* $\{g, b\}$ *has outdegree two.*

Both quadrangulations and pseudo-triangulations admit a binary labelling. Special properties of this labelling for quadrangulations are explained in the next section.

## 3 The binary labelling for quadrangulations

Let a quadrangulation $Q$ be given such that the outer face of $Q$ contains four vertices. For a binary labelling of $Q$ we also require the following properties:

5. Each face has two adjacent 0-labels and two adjacent 1-labels, i.e. it is labelled $(0, 0, 1, 1)$.

6. The edges incident to $g$ are labelled $0, 1 - 0, 0$ where the 1 stands on the right side of the edge (seen from vertex $g$).

Figure 1 shows a binary labelling.

**Theorem 1** *Every quadrangulation with four vertices on its outer face admits a binary labelling.*

**Proof.** We use induction on the number of vertices $|V|$ of a quadrangulation $Q$. If $|V| = 4$ then a binary labelling exists, as shown in Figure 3 (left). For the induction step we distinguish two cases.

First, assume that $Q$ contains an interior vertex $v$ of degree two. Removal of $v$ and its two incident edges yields a quadrangulation $Q'$. By induction, $Q'$ admits a binary labelling. Reinserting $v$ and its incident edges into $Q'$ maintains the binary labelling, as shown in Figure 3 (right).

We now assume that $Q$ contains no interior vertex of degree two. In this case, there exists a face incident to the special vertex $g$ which can be *contracted* towards $g$. A face $q$ incident to $g$ is contractible if it does not contain the other special vertex $b$. A contraction of $q = \{e', e, f, f'\}$, where $\{e', e, f, f'\}$ are the edges of $q$ in cyclic order and $e'$ and $f'$ are incident to $g$ and $e$ and $f$ are incident to the vertex $p$ opposite to $g$, identifies $e$ with $e'$, $f$ with $f'$ and $p$ with $g$. It can be interpreted as a continuous movement of $p$ and its incident edges to $g$.

A contraction of $q$ yields a quadrangulation which by induction admits a binary labelling. In particular, the edges $e'$ and $f'$ are labelled $0, 1 - 0, 0$ towards $g$ (Property 6 of the binary labelling). Now, we reverse the contraction. This operation maintains the binary labelling outside of the face $q$. It remains to label the angles inside $q$. The vertex rule for the special vertex $g$ requires that the angle incident to $g$ is labelled with 0. Labelling the angle formed by $e$ and $e'$ with 1 maintains Property 6 for $e'$ and guarantees the vertex rule for this vertex. Observe that now the edge $e$ is labelled with endpoint $1, 1$. Hence, the angle at $p$ inside $q$ has to be labelled with 1 to guarantee the edge rule for $e$. Note that all other labels at $p$ are 0. Thus, labelling $p$ with 1 also ensures the vertex rule for $p$. Finally, labelling the angle formed by $f$ and $f'$ with 0 guarantees the edge rule for $f$ and $f'$; here, we again use Property 6. Observe that the vertex rule is satisfied for this vertex; and $q$ satisfies the face rule.

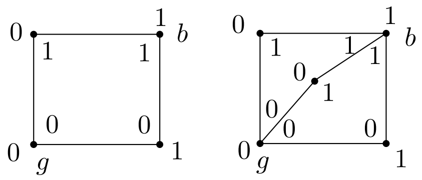


Figure 3: The basis of the induction and inserting a vertex of degree two.

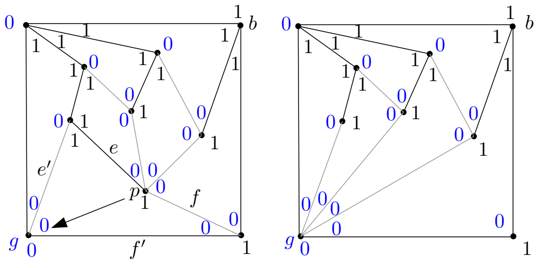


Figure 4: Contracting a quadrangle to the special vertex $g$.

Figure 4 shows a contraction of a quadrangle and the resulting binary labelling.

□

## 3.1 The two regions of a vertex

Analogous versions of the following results have been given by Schnyder [14] for triangulations and by Felsner [4] for 3-connected planar graphs. The proofs are omitted in this abstract.

We denote with $T_0$ the union of the edges which are oriented towards $0, 0$. We denote with $T_1$ the union of the edges which are oriented towards $1, 1$.

**Lemma 2** *$T_i, i \in \{0, 1\}$, is a directed tree rooted at $g$, respectively $b$.*

**Lemma 3** *There is no directed cycle in $T_0 \cup T_1^{-1}$. There is no directed cycle in $T_1 \cup T_0^{-1}$.*

For each interior vertex $v$ and $i \in \{0, 1\}$, we define the $i$-path $P_i(v)$ starting at $v$ as the path in $T_i$ from $v$ to the root of $T_i$. $P_0(v)$ and $P_1(v)$ have $v$ as only common vertex. Therefore $P_0(v)$ and $P_1(v)$ divide the quadrangulation into two regions $R_0(v)$ and $R_1(v)$.

**Lemma 4** *For any two distinct interior vertices $u$ and $v$ and for $i \in \{0, 1\}$ there holds the implication $u \in R_i(v) \Rightarrow R_i(u) \subset R_i(v)$.*

The following result has been proved with a different method in [1].

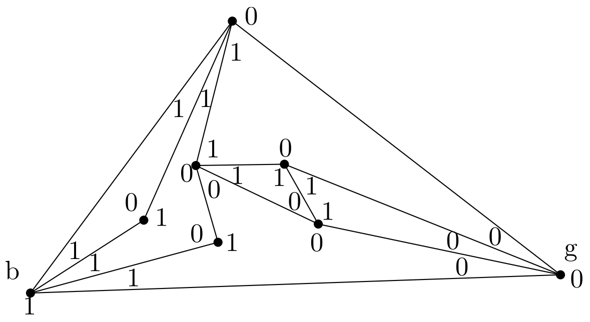


Figure 5: A binary labelling for a pointed pseudo-triangulation.

**Lemma 5** *Every quadrangulation can be decomposed into two edge-disjoint spanning trees by duplicating edges of the outer face.*

Let $Q$ be a quadrangulation and let $\widetilde{Q}$ be its dual graph. Then the "dual" of the two spanning trees $T_i$ for $i = 1, 2$ are two spanning trees decomposing $\widetilde{Q}$.

## 4 A binary labelling for plane Laman graphs

We consider an analogous binary labelling for plane Laman graphs. Now, the special vertices $b$ and $g$ are adjacent convex hull vertices. Thus there is one edge which does not satisfy the edge rule.

Figure 5 shows a binary labelling for a pointed pseudo-triangulation.

**Theorem 6** *Every plane Laman graph with three vertices on its outer face admits a binary labelling.*

**Sketch of Proof.** Our proof of the binary labelling is based on the *Henneberg construction*[17, 8, 16, 11]. A Laman graph can be constructed, starting from a triangle, by a sequence of vertex insertions of the following types (see Figure 6)

1. Add a degree-two vertex (Henneberg I step)
2. Place a vertex on an existing edge and connect it to a third vertex (Henneberg II step).

The binary labelling can be proved inductively. This amounts to showing that a Henneberg step maintains the labelling. To guarantee planarity, we make use of a lemma from Haas et al. [8]. We omit the details. □

It is well known that a Laman graph can be decomposed into two trees [17]. These trees can be obtained via the Henneberg construction, as indicated in Figure 6. The new vertex either is a leaf in both trees (Henneberg I step) or in one tree (Henneberg II step).

Although the binary labelling is based on the Henneberg construction too, it does not always give a decomposition of the graph into two trees; see Figure 7

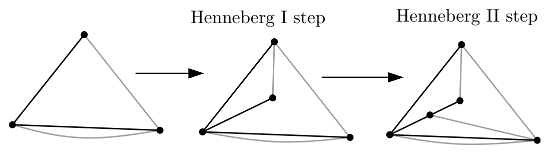


Figure 6: Constructing a decomposition into two spanning trees via Henneberg steps.

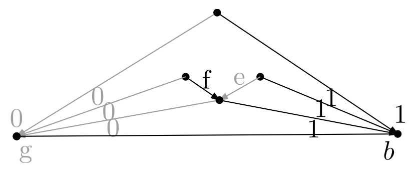


Figure 7: The binary labelling for plane Laman graphs does not induce a decomposition into two trees.

for a simple example: The angles around the special vertex $g$, respectively $b$, are labelled with 0, respectively 1. All edges incident to $g$ are gray, all edges incident to $b$ are black. Thus, if there is a decomposition of the edge set into two trees, then the edge $f$ has to be black and oriented towards $b$, and the edge $e$ has to be gray and oriented towards $g$. But then, the angle formed by $e$ and $f$ has to labelled with 1 and with 0, contradicting to properties of the binary labelling.

We remark that variants of the binary labelling hold for plane Laman graphs containing more than three vertices on the outer face.

## 5 Open problems

A main application of the Schnyder labelling for triangulations is a straight-line embedding of a triangulation on an $n-2$ by $n-2$ grid. Biedl and Brandenburg [2] recently showed that every planar bipartite graph has a straight-line embedding on a grid of size $\lfloor \frac{n}{2} \rfloor$ by $\lceil \frac{n}{2} \rceil - 1$. What is the corresponding grid size for planar Laman graphs? We did not succeed in applying the binary labelling. Another related question, posed by Haas et al. [8], is the following: Can every planar Laman graph be embedded as a pseudo-triangulation on a grid of small size?

## References

[1] O. Aichholzer, F. Aurenhammer, P. Gonzalez-Nava, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, S. Ray, B. Vogtenhuber. Matching edges and faces in polygonal partitions. *In Proc. 17th Canadian Conference on Computational Geometry*, 123–126, Windsor, 2005.

[2] T. Biedl, F. Brandenburg. Drawing planar bipartite graphs with small area. *In Proc. 17th Canadian Conference on Computational Geometry*, 105–108, Windsor, 2005.

[3] Z. Fekete, T. Jordán, W. Whiteley. An Inductive Construction for Plane Laman Graphs via Vertex Splitting. *European Symposium on Algorithms* 299–310, Bergen, 2004.

[4] S. Felsner. Convex Drawings of Planar Graphs and the Order Dimension of 3-Polytopes. *Order* 18:19–37, 2001

[5] H. de Fraysseix, P.Ossona de Mendez. On topological aspects of orientations. *Discrete Mathematics* 229:57–72, 2001

[6] H. de Fraysseix, P. Ossona de Mendez, J. Pach. A left-first search algorithm for planar graphs. *Discrete Computational Geometry* 13:459–468, 1995.

[7] F. Harary. Graph Theory. *Addison-Wesley*, Reading, 1969

[8] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, W. Whiteley. Planar minimally rigid graphs and pseudo-triangulations. *Computational Geometry, Theory and Applications* 31:31–61, 2005.

[9] G. Laman. On Graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics* 4:331-340, 1970.

[10] A.S. Lladó, S.C. López Masip. Decompositions of graphs with a given tree. *Actas III Jornadas de Matemática Discreta y Algorítmica*, 204-211, Sevilla, 2002.

[11] D. Orden. Ph. D. thesis: Two problems in geometric combinatorics: Efficient triangulations of the hypercube; Planar graphs and rigidity. Universidad de Cantabria, 2003.

[12] V. Petrović. Decomposition of some planar graphs into trees. *Discrete Mathematics* 150:449–451, 1996.

[13] G. Ringel. Two Trees in Maximal Planar Bipartite Graphs. *Journal of Graph Theory* 17:755–758, 1993.

[14] W. Schnyder. Embedding planar graphs on the grid. *In Proc. 1st ACM/SIAM Symposium on Discrete Algorithms* 138–148, 1990.

[15] D. Souvaine, C. Tóth. A vertex-face assignment for plane graphs. *In Proc. 17th Canadian Conference on Computational Geometry*, 138–141, Windsor, 2005.

[16] I. Streinu. A Combinatorial Approach to Planar Non-Colliding Robot Arm Motion Planning. *In Proc. 41st Symposium on Foundations of Computer Science* 443-453, 2000.

[17] T.-S. Tay, W. Whiteley. Generating isostatic frameworks. *StructuralTopology* 11:21-69, 1985.

# Helly-Type Theorems for Line Transversals to Disjoint Unit Balls

Otfried Cheong[*] Xavier Goaoc[†] Andreas Holmsen[‡] Sylvain Petitjean[§]

**Abstract**

We prove Helly-type theorems for line transversals to disjoint unit balls in $\mathbb{R}^d$. In particular, we show that a family of $n \geqslant 2d$ disjoint unit balls in $\mathbb{R}^d$ has a line transversal if, for some ordering $\prec$ of the balls, every subfamily of $2d$ balls admits a line transversal consistent with $\prec$. We also prove that a family of $n \geqslant 4d-1$ disjoint unit balls in $\mathbb{R}^d$ admits a line transversal if every subfamily of size $4d-1$ admits a transversal.

Helly's celebrated theorem, published in 1923, states that a finite family of convex sets in $\mathbb{R}^d$ has non-empty intersection if and only if every subfamily of size at most $d+1$ has non-empty intersection. Subsequent results of similar flavor (that is, if every subset of size $k$ of a set $\mathcal{S}$ has property $\mathcal{P}$ then $\mathcal{S}$ has property $\mathcal{P}$) have been called *Helly-type theorems* and the minimal such $k$ is known as the associated *Helly number*. Helly-type theorems and tight bounds on Helly numbers have been the object of active research in combinatorial geometry. In this paper, we investigate Helly-type theorems for the existence of line transversals to a family of objects, i.e. lines that intersect every member of the family.

**History.** The earliest Helly-type theorems in geometric transversal theory appeared about five decades ago. In 1957, Hadwiger [10] showed that an ordered family $\mathcal{S}$ of compact convex figures in the plane admits a line transversal if every triple admits a line transversal compatible with the ordering. (Note that a line transversal to $\mathcal{S}$ may not respect the ordering on $\mathcal{S}$; to prove the existence of a line transversal that respects the ordering on $\mathcal{S}$ one needs the assumption that any *four* admits an order-respecting line transversal.) In what follows, we shall talk about a Hadwiger-type theorem when the family of objects under consideration is ordered.

The same year, L. Danzer [4] proved the following result concerning families of pairwise disjoint unit discs in the plane: if such a family consists of at least 5 discs, and if any 5 of these discs are met by some line, then there exists a line meeting all the discs of the family. This answered a question of Hadwiger [8], who gave an example (5 circles, almost touching and with centers forming a regular pentagon) which shows that 5 cannot be replaced by 4. Grünbaum [6] showed that the same result holds if "unit disc" is replaced by "unit square", and conjectured that the result holds for families of disjoint translates of any compact convex set in the plane. This long-standing conjecture was finally proved by Tverberg [14]. A weaker form of the conjecture which assumed 128 instead of 5 had been established earlier by Katchalski [13].

In three dimensions, neither Hadwiger nor Helly-type theorems exist for line transversals to general convex objects, not even for translates of a convex compact set [12]. However, Hadwiger [9] proved a Helly-type theorem for line transversals to "thinly distributed" disjoint balls in dimension $d$ with Helly number $d^2$. A family of balls is thinly distributed if the distance between any two balls is at least the sum of their radii. Grünbaum [7] improved this Helly number to $2d-1$ using the topological Helly theorem. For the special case of unit balls in three dimensions—but without any additional assumption on their distribution—Holmsen et al. [11] showed a Hadwiger-type theorem with constant 12, and a Helly-type theorem with constant 46. These constants were later improved to 9 and 18 by Cheong et al. [3].

We refer the reader to the recent survey by Wenger [15] for a broader discussion of geometric transversal theory.

**Our results.** In this paper we prove Helly-type theorems for line transversals to families of *pairwise-inflatable* balls in $\mathbb{R}^d$. A family $\mathcal{F}$ of balls in $\mathbb{R}^d$ is called pairwise-inflatable if for every pair of balls $B_1, B_2 \in \mathcal{F}$ we have $\gamma^2 > 2(r_1^2 + r_2^2)$, where $r_i$ is the radius of $B_i$, and $\gamma$ is the distance between their centers. A family of disjoint unit balls is pairwise-inflatable, and so is a family of balls that is "thinly distributed" in Hadwiger's sense. Pairwise-inflatable families of balls are not only more general than families of disjoint congruent balls but allow to generalize most of our proofs obtained in three or four dimensions to arbitrary dimension; the key property, which we prove in this paper, is that the set of pairwise-

[*]Division of Computer Science, KAIST, Daejeon, South Korea. Email: otfried@kaist.ac.kr. Otfried Cheong was supported by LG Electronics.

[†]LORIA–INRIA Lorraine, Nancy, France. Email: goaoc@loria.fr.

[‡]Department of Mathematics, University of Bergen, Bergen, Norway. Email: andreash@mi.uib.no

[§]LORIA–CNRS, Nancy, France. Email: petitjea@loria.fr.

inflatable families is closed under intersection with affine subspaces, unlike the set of families of disjoint congruent balls.

An order-respecting line transversal to a subset of an ordered family is a line transversal that respects the order induced by the family on that subset. An ordered family $\mathcal{F}$ of pairwise-inflatable balls is said to have property $(OR)T$ if it admits a (order-respecting) line transversal. If every $k$ or fewer members of $\mathcal{F}$ admit a (order-respecting) line transversal then $\mathcal{F}$ is said to have property $(OR)T(k)$. Our first main result requires that the line transversals to the subfamilies induce consistent orderings:

**Proposition 1** *For any ordered family of pairwise-inflatable balls in $\mathbb{R}^d$, $ORT(2d)$ implies $T$ and $ORT(2d+1)$ implies $ORT$.*

We then remove the condition on the ordering at the cost of increasing the Helly number to $4d-1$ and restricting ourselves to disjoint unit balls:

**Proposition 2** *For any family of disjoint unit balls in $\mathbb{R}^d$, $T(4d-1)$ implies $T$.*

Our results are thus both qualitative and quantitative: we generalize Danzer's result to arbitrary dimension and prove that the Helly number grows at most linearly with the dimension. We build on the work of Holmsen et al. [11] who obtained results similar to Propositions 1 and 2 for disjoint unit balls in three dimensions, albeit with larger bounds on Helly numbers (12 and 46 instead of 6 and 11, respectively). A previous version of this paper, also restricted to disjoint unit balls in three dimensions, appeared in the Symposium on Computational Geometry 2005 [1].

**Approach.** To prove Proposition 1, we start with a family of balls having property $ORT(2d)$ and continuously shrink them until that property no longer holds, following Hadwiger's approach [10]. Before the set of order-respecting line transversals to a $2d$-tuple of balls disappears (i) it first reduces to a single line and (ii) this line is an isolated line transversal to $2d-1$ of the balls. That line has then to be a line transversal to the whole family and Proposition 1 follows; considerations on geometric permutations yield Proposition 2.

Proving the properties (i) and (ii) mentioned above is elementary in the plane but requires considerably more work in higher dimension. For a sequence $\mathcal{F}$, let $\mathcal{K}(\mathcal{F}) \subset \mathbb{S}^{d-1}$ denote the set of directions of line transversals to $\mathcal{F}$. Our proofs rely on the following proposition:

**Proposition 3** *The directions of order-respecting line transversals to a family of pairwise-inflatable balls in $\mathbb{R}^d$ form a strictly convex subset of $\mathbb{S}^{d-1}$.*

This directly implies property (i) and yields that order-respecting line transversals form a contractible set in line space. From there, a well-known topological analogue of Helly's theorem leads to a weaker version of Proposition 1 sufficient to prove property (ii), namely:

**Proposition 4** *If a line $\ell$ is an isolated line transversal to a sequence $\mathcal{F}$ of $n \geqslant 2d$ pairwise-inflatable balls in $\mathbb{R}^d$ then there exists a subsequence $\mathcal{F}' \subset \mathcal{F}$ of size $2d-1$ such that $\ell$ is an isolated line transversal to $\mathcal{F}'$.*

For the proofs, omitted in this extended abstract, we refer the reader to the full version [2].

**Open problems.** We conclude by a few open problems suggested by our results.

**Problem 1** *What is the maximum number of geometric permutations of pairwise-inflatable balls in $\mathbb{R}^d$?*

A geometric permutation of a collection of disjoint convex sets is an ordering of these sets that can be realized by a line transversal. To prove Proposition 2 we use the fact that the number of geometric permutations of $n$ disjoint balls in $\mathbb{R}^d$ is at most 3 if the balls have equal radii [3]. If the ratio

$$\frac{\text{largest radius}}{\text{smallest radius}}$$

is not bounded independently of $n$ then the number of geometric permutations is known to be $\Theta(n^{d-1})$ [16].

**Problem 2** *For which classes of objects is the cone of directions $\mathcal{K}(A_1, \ldots, A_n)$ convex, or at least contractible?*

Our proof of convexity for the cone of directions of balls collapses for balls that are not pairwise-inflatable. In fact, the set $Q^F_{AB}$ is not necessarily convex if $B$ is much smaller than $A$ but very close to it.

**Problem 3** *For which classes of objects is the set of order-respecting line transversals always connected?*

Our proof of Proposition 1 follows from (i) a bounded pinning number and (ii) the fact that as the set of order-respecting line transversals to a sequence disappears it first reduces to a single line. For strictly convex objects, property (ii) follows from the connectivity of the set of order-respecting transversals. Surprisingly, it is an open question whether this set is connected for even 4 disjoint balls in $\mathbb{R}^3$, whereas it is known to be connected for any triple of disjoint convex objects [5, Lemma 74]. We conjecture that general convex sets in $\mathbb{R}^d$ have a bounded pinning number. Thus, understanding how general this connectivity property is would provide insight in how general the example of Holmsen and Matousek [12], convex sets whose translates do not admit a Hadwiger theorem, actually is. Of course, a positive answer to Problem 2 for a particular family of convex sets implies a positive answer to Problem 3 for that family as well.

**Problem 4** *Is the pinning number of disjoint unit balls in $\mathbb{R}^d$ equal to $2d-1$?*

Surprisingly, the only known lower bound on the Helly number is the construction done by Hadwiger fifty years ago. Note that the bound in our Hadwiger theorem has to be higher than the pinning number of the corresponding family and one can therefore look for a lower bound on the pinning number. Intuitively, considerations on the dimension suggest that the pinning number in dimension $d$ cannot be less than $2d-1$, the dimension of the underlying line space being $2d-2$.

## Acknowledgments

We thank Gregory Ginot, Günter Rote and Guillaume Batog for helpful discussions.

## References

[1] O. Cheong, X. Goaoc, and A. Holmsen. Hadwiger and Helly-type theorems for disjoint unit spheres in $\mathbb{R}^3$. In *Proc. 20th Ann. Symp. on Computational Geometry*, pages 10–15. 2005.

[2] O. Cheong, X. Goaoc, A. Holmsen, and S. Petitjean. Helly-type theorems for line transversals to disjoint unit balls. Available from `http://tclab.kaist.ac.kr/~otfried/Papers/cghp-httlt.pdf`

[3] O. Cheong, X. Goaoc, and H.-S. Na. Geometric permutations of disjoint unit spheres. *Comput. Geom. Theory Appl.*, 2005. In press.

[4] L. Danzer. Über ein Problem aus der kombinatorischen Geometrie. *Arch. der Math*, 1957.

[5] X. Goaoc. *Structures de visibilité globales : tailles, calculs et dégénérescences.* Thèse d'université, Université Nancy 2, May 2004.

[6] B. Grünbaum. On common transversals. *Arch. Math.*, IX:465–469, 1958.

[7] B. Grünbaum. Common transversals for families of sets. *J. London Math. Soc.*, 35:408–416, 1960.

[8] H. Hadwiger. Ungelöste Probleme, No. 7. *Elem. Math.*, 1955.

[9] H. Hadwiger. *Wiskundige Opgaven*, pages 27–29, 1957.

[10] H. Hadwiger. Über Eibereiche mit gemeinsamer Treffgeraden. *Portugal Math.*, 6:23–29, 1957.

[11] A. Holmsen, M. Katchalski, and T. Lewis. A Helly-type theorem for line transversals to disjoint unit balls. *Discrete Comput. Geom.*, 29:595–602, 2003.

[12] A. Holmsen and J. Matoušek. No Helly theorem for stabbing translates by lines in $\mathbb{R}^d$. *Discrete Comput. Geom.*, 31:405–410, 2004.

[13] M. Katchalski. A conjecture of Grünbaum on common transversals. *Math. Scand.*, 59(2):192–198, 1986.

[14] H. Tverberg. Proof of Grünbaum's conjecture on common transversals for translates. *Discrete & Comput. Geom.*, 4(3):191–203, 1989.

[15] R. Wenger. Helly-type theorems and geometric transversals. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 4, pages 73–96. CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.

[16] Y. Zhou and S. Suri. Geometric permutations of balls with bounded size disparity. *Comput. Geom. Theory Appl.*, 26:3–20, 2003.

# Geometric realization of a projective triangulation with one face removed

C. Paul Bonnington* Atsuhiro Nakamoto† Kyoji Ohba‡

## 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 our talk, we shall prove that every triangulation $G$ on the projective plane has a face $f$ such that the triangulation of the Möbius band obtained from $G$ by removing the interior of $f$ has a geometric realization.

## 1 Introduction

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 $G$ be a map on a surface $F^2$. A *geometric realization* of $G$ is an embedding of $F^2$ into a Euclidian 3-space $\mathbb{R}^3$ with no self-intersection such that each face of $G$ is a flat polygon. That is, a geometric realization of $G$ is to express $G$ as a polytope $P(G)$ in $\mathbb{R}^3$ such that $P(G)$ is homeomorphic to $F^2$, and that the 1-skeleton of $P(G)$ is homeomorphic to the graph of $G$. Note that we do not require the convexity of $P(G)$.

Steinitz's theorem states that a spherical map has a geometric realization if and only if its graph is 3-connected [7]. 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 [5], 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].

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 a 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.

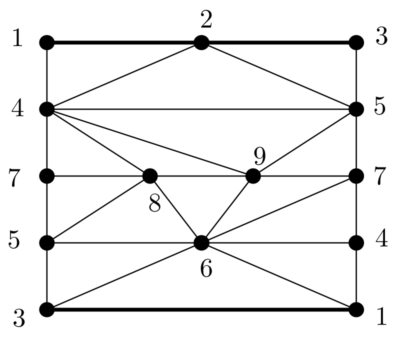


Figure 1: A Möbius triangulation with no geometric realization

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.

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 $f$.

The following is our main theorem.

**Theorem 1** *Every projective triangulation $G$ has a face $f$ such that the Möbius triangulation $G-f$ has a geometric realization.*

As far as we know, the current result seems to be the first affirmative result for geometric realizations of maps on nonorientable surfaces, since Brehm found the counterexample shown in Figure 1.

## 2 Sketch of the proof

In this section, we briefly explain our graph-theoretical proof of the theorem.

Let $M$ be a map on a surface $F^2$ and let $e$ be an edge of $M$. *Contraction* of $e$ in $M$ is to removed $e$ and

*Department of Mathematics, University of Auckland, Auckland, New Zealand Email: p.bonnington@auckland.ac.nz

†Department of Mathematics, Yokohama National University, Yokohama 240-8501, Japan Email: nakamoto@edhs.ynu.ac.jp

‡Yonago National College of Technology, Yonago 683-8502, Japan Email: ooba@yonago-k.ac.jp

identify the two endpoints of $e$. (The inverse operation of contraction of an edge is called a *splitting of a vertex.*) If this yields a face bounded by a 2-cycle, then we replace the two parallel edges with a single edge. The contraction of an edge $e$ is allowed only if the graph, say $H$, obtained from $M$ by the contraction of $e$ is simple. In this case, we say that $e$ is *contractible*, and that $M$ is *contractible* to $H$. We say that $M$ is *irreducible* if $M$ has no contractible edge.

Barnette [2] proved that the projective plane admits exactly two irreducible triangulations, which are the complete graph $K_6$ with six vertices and $K_4+\overline{K_3}$ (i.e., the quadrangulation by $K_4$ with each face subdivided by a single vertex), which are shown in the left-hand side in Figures 2 and 3, respectively. In particular, the latter contains a quadrangulation by $K_4$ with vertex set $\{1, 2, 3, 4\}$, called a *$K_4$-quadragulation*, which will play an important role in our proof.

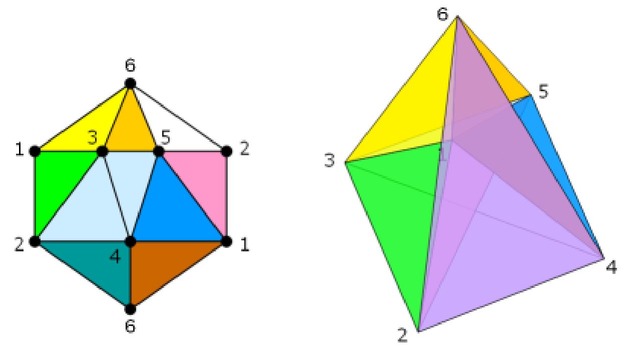


Figure 2: A geometric realization of $K_6$ minus face 256

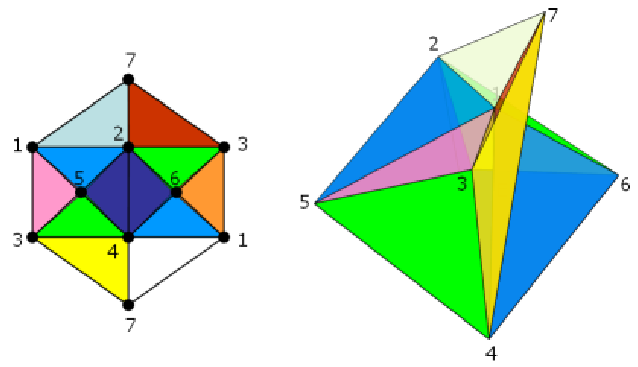


Figure 3: A geometric realization of $K_4 + \overline{K_2}$ minus face 147

**Lemma 2** *Each of the two irreducible projective triangulation with one face removed has a geometric realization.*

The right-hand side of Figures 2 and 3 show geometric realizations of the two irreducible triangulations with one face removed, respectively. Note that each of the two triangulations is symmetric, the map with any one face removed has a geometric realization.

In order to prove Theorem 1, it is difficult to use induction on the number of vertices, though the first step of induction is verified by Lemma 1. Therefore, we introduce the following lemma to classify the set of all projective triangulations into two classes, which contain two irreducible projective triangulations independently.

**Lemma 3** *Let $G$ be a projective triangulation. Then $G$ is contractible to $K_6$ if and only if $G$ does not contain a $K_4$-quadrangulation.*

Throughout the proof, we use Menger's Theorem many times, which is well-known in graph theory and states that for a graph $G$ and its two disjoint vertex-sets $A, B$ with cardinality $k$, there are $k$ disjoint paths joining $A$ and $B$, unless $G$ has a cut set $X$ with cardinality at most $k-1$ separating $A$ and $B$.

We first consider a projective triangulation $G$ with a $K_4$-quadrangulation as a subgraph. Applying Menger's Theorem suitably, we can easily find a subdivision of $K_4+\overline{K_3}$. Since $K_4+\overline{K_3}$ itself satisfies Theorem 1 by Lemma 1, it is not difficult to construct a geometric realization of $G$ with one face removed such that each path of $G$ corresponding an edge of $K_4+\overline{K_3}$ is a straight-line segment, and that each 2-cell region of $G$ corresponding a face of $K_4+\overline{K_3}$ is a flat triangle.

By Lemma 2, if a projective triangulation has no $K_4$-quadrangulationas a subgraph, then it is contractible to $K_6$. In other words, in this case, $G$ has a $K_6$*-minor* as a subgraph, which is a map transformed into $K_6$ by a sequence of contracting and deleting edges.

Let $K_5$ denote a Möbius triangulation obtained from the triangulation $K_6$ by removing the 2-cell region consisting of five triangular faces incident to a single vertex. A $K_5$*-minor* is a map on the Möbius band transformed into $K_5$ by a sequence of contracting and deleting edges. Observe that there are two ways to split a vertex of $K_5$, depending on whether a new edge arisen by the splitting lies on the boundary of the Möbius band, or not. The former is called a *boundary splitting*, and the latter an *innr splitting.* Hence there are several homeomorphism-classes of the $K_5$-minors.

**Lemma 4** *A $K_5$-minor has a geometric realization.*

For example, Figure 4 shows a $K_5$-minor obtained by five boundary splittings and its geometric realization, in which we can see that contracting $v_i$ and $v_i'$ for each $i$ yields $K_5$.

Finally, we have to put in a 2-cell region, say $R$, with one face removed to the body of the geometric realization of a $K_5$-minor constructed in Lemma 4. By using Menger's Theorem carefully, we can take

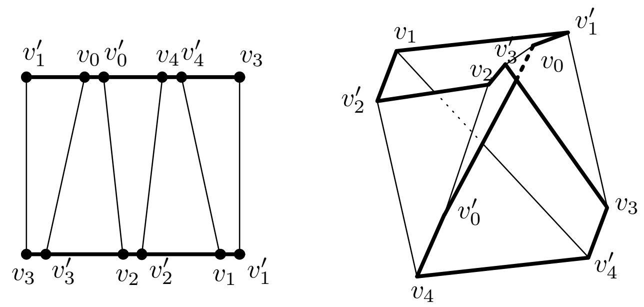


Figure 4: A $K_5$-minor obtained by boundary splittings of all five vertices and its geometric realization

an inner vertex $v$ in $R$ which has disjoint five paths $Q_0, \ldots, Q_4$ to five corners $v_0, \ldots, v_4$, as shown in Figure 5, for example. The argument is complicated, and hence we omit the details.

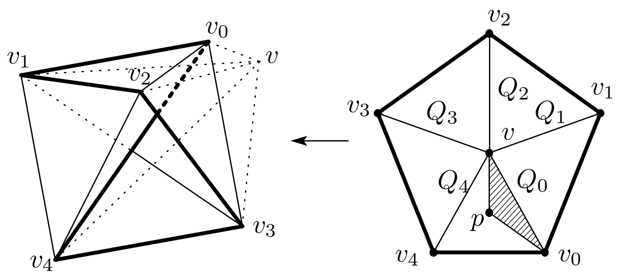


Figure 5: Putting in $R$ with one face removed to $K_5$.

## 3 Conclusion and Conjecture

Our problem is as follows:

Problem. Let $M$ be a map on the projective plane and let $f$ be a face of $M$. Does $M - f$ have a geometric realization?

Our main result is Theorem 1, that is, if $M$ is a triangulation and $f$ is some face of $M$, the answer for the above problem is "yes". Note that in Theorem 1, $f$ cannot be chosen arbitrarily since there is a counterexample by Brehm shown in Figure 1. Moreover, we do not know whether the restriction to be a triangulation in Theorem 1 is actually needed. Therefore, it will be nice to consider the above problem when $M$ is a Petersen graph on the projective plane with each face pentagon, which is a surface dual of $K_6$.

Why does the Brehm's counterexample have no geometric realization? A key of the proof is that in every spatial embedding of the map shown in Figure 1, the two disjoint 3-cycles 123 and 456 have a linking number at least 2. (See [6] for the definition of the linking number.) However, any two 3-cycles with straight segments have linking number at most one, a contradiction. Therefore, we have the following observation:

**Observation 1** *If a Möbius triangulation $G$ has a boundary cycle $C$ of length 3 and a 3-cycle $C'$ disjoint from $C$ which forms an annular region with $C'$, then $G$ never has a geometric realization.*

A graph $G$ is said to be *cyclically $k$-connected* if $G$ has no separating set $S \subset V(G)$ of $G$ with $|S| \leq k - 1$ such that each connected component of $G - S$ has a cycle. If we assume the cyclically 4-connectedness of a Möbius triangulation, we can avoid the situation described in Observation 1. So we conjecture the following, which will enable us to prove Theorem 1 easily.

**Conjecture 1** *Let $G$ be a projective triangulation. Then $G$ is cyclically 4-connected if and only if $G - f$ has a geometric realization for any face $f$ of $G$.*

Clearly, the above characterizes a geometrically realizable Möbius triangulation to be cyclically 4-connected. This answers the question given in [1].

In the previous section, we have briefly explained how to construct a geometric realization of a projective triangulation with one face removed. Of course, we needed an observation from a geometrical point of view. However, we feel that a graph-theoretical method is more essential, that is, we will need to find a geometrically realizable specific subgraph in an arbitrarily given projective triangulation.

## References

[1] D. Archdeacon, C.P. Bonnington and J.A. Ellis-Monanghan, How to exhibit toroidal maps in space, preprint.

[2] D.W. Barnette, Generating triangulations of the projective plane, *J. Combin. Theory Ser. B* **33** (1982), 222–230.

[3] J. Bokowski and A. Guedes de Oliveira, On the generation of oriented matorids, *Discrete Comput. Geom.* **24** (2004), 197–208.

[4] U. Brehm, A nonpolyhedral triangulated Möbius strip, *Proc. Amer. Math. Soc.* **89** (1983), 519–522.

[5] B. Grünbaum, "Convex polytopes", *Pure and Applied Mathematics* Vol. 16, Interscience-Wiley, New York, 1967.

[6] D. Rolfsen, *Knot and links*, Math. Lecture Series 7, Publish or Perish, Berkeley, Calif., 1976.

[7] E. Steinitz, Polyeder un Raumeinteilunger, *Enzykl. Math. Wiss.* Vol. 3, Teil 3A612 (1922), 1–139.

# Splitting (Complicated) Surfaces Is Hard

Erin W. Chambers[*] Éric Colin de Verdière[†] Jeff Erickson[‡] Francis Lazarus[§] Kim Whittlesey[¶]

**Abstract**

Let $\mathcal{M}$ be an orientable surface without boundary. A cycle on $\mathcal{M}$ is *splitting* if it has no self-intersections and it partitions $\mathcal{M}$ into two components, neither homeomorphic to a disk. In other words, splitting cycles are simple, separating, and non-contractible. We prove that finding the shortest splitting cycle on a combinatorial surface is NP-hard but fixed-parameter tractable with respect to the surface genus. Specifically, we describe an algorithm to compute the shortest splitting cycle in $g^{O(g)} n \log n$ time.

## 1 Introduction

Optimization problems on surfaces in the fields of computational topology and topological graph theory have received much attention in the past few years. Such problems are usually set in the *combinatorial surface* model. A combinatorial surface is a graph $G(\mathcal{M})$ embedded on a surface $\mathcal{M}$ that cuts $\mathcal{M}$ into topological disks. Curves on this surface are required to be walks on $G(\mathcal{M})$, and edges of $G(\mathcal{M})$ have positive weights, allowing to measure the length of a curve.

Many of these problems can be seen as the computation of a shortest cycle with some prescribed topological property, such as non-contractibility. When the set of cycles with the desired property satisfies the so-called *3-path condition*, a generic algorithm of Mohar and Thomassen finds a shortest such cycle in $O(n^3)$ time [11, Sect. 4.3]. For instance, the sets of non-separating and non-contractible cycles satisfy the 3-path condition.

In this paper, we study the following optimization problem: Given an orientable 2-manifold $\mathcal{M}$ with genus $g \geq 2$ without boundary, find a shortest simple non-contractible cycle that separates $\mathcal{M}$. For simplicity, we will call a simple non-contractible separating cycle a *splitting* cycle. The set of splitting cycles does not satisfy the 3-path property. Removing a splitting cycle from any surface leaves two surfaces, each of genus at least one and with one boundary cycle.

We prove that finding the shortest splitting cycle on a combinatorial surface is NP-hard but fixed-parameter tractable with respect to the surface genus. Specifically, we describe an algorithm to compute the shortest splitting cycle in $g^{O(g)} n \log n$ time.

## 2 Topological Background

### 2.1 Curves on surfaces

We rely on several notions from combinatorial topology. In particular, we use the standard definitions for a *compact, orientable, and connected surface* $\mathcal{M}$, its *genus*, a *path*, a *loop*, or a *cycle* on $\mathcal{M}$, *homotopy with or without basepoint*. See also Hatcher [8] or previous papers [4, 2] for more details. We say that two loops are disjoint if they intersect only at their common basepoint. We say that a cycle *splits* a surface $\mathcal{M}$ if it is simple, non-contractible, and separating.

### 2.2 Systems of loops

If $L$ is a set of pairwise disjoint simple loops, $\mathcal{M} \setminus L$ denotes the surface with boundary obtained by *cutting* $\mathcal{M}$ along the loops in $L$. A *system of loops* on $\mathcal{M}$ is a set of pairwise disjoint simple loops $L$ such that $\mathcal{M} \setminus L$ is a topological disk. Any system of loops contains exactly $2g$ loops. $\mathcal{M} \setminus L$ is a $4g$-gon where the loops appear in pairs on its boundary, and is called the *polygonal schema* associated to $L$.

### 2.3 Combinatorial and cross-metric surfaces

Like most earlier related results [1, 3, 4, 5, 6, 10], we state and prove our results in the *combinatorial surface* model. A combinatorial surface is an abstract surface $\mathcal{M}$ together with a weighted undirected graph $G = G(\mathcal{M})$, embedded on $\mathcal{M}$ so that each open face is a disk. In this model, the only allowed paths are walks in $G$; the length of a path is the sum of the weights of the edges traversed by the path, counted with multiplicity. A path is *simple* if it can be slightly

[*]Department of Computer Science, University of Illinois, `erinwolf@uiuc.edu`. Research partially supported by an NSF Graduate Research Fellowship and by NSF grant DMS-0528086.

[†]CNRS, Laboratoire d'informatique de l'École normale supérieure, Paris, France, `Eric.Colin.de.Verdiere@ens.fr`

[‡]Department of Computer Science, University of Illinois, `jeffe@cs.uiuc.edu`. Research partially supported by NSF grants CCR-0093348, CCR-0219594, and DMS-0528086.

[§]CNRS, Laboratoire des Images et des Signaux, Grenoble, France, `Francis.Lazarus@lis.inpg.fr`

[¶]Department of Mathematics, University of Illinois, `kwhittle@math.uiuc.edu`

deformed so as to become a simple path on the surface. The *complexity* of a combinatorial surface is the total number of vertices, edges, and faces of $G$.

It is often more convenient to work in an equivalent dual formulation of this model introduced by Colin de Verdière and Erickson [2]. A *cross-metric surface* is also an abstract surface $\mathcal{M}$ together with an undirected weighted graph $G^* = G^*(\mathcal{M})$, embedded so that every open face is a disk. However, now we consider only *regular* paths and cycles on $\mathcal{M}$, which intersect the edges of $G^*$ only transversely and away from the vertices. The *length* of a regular curve $p$ is defined to be the sum of the weights of the dual edges that $p$ *crosses*, counted with multiplicity. See [2] for further discussion of these two models.

## 3 NP-hardness

**Theorem 1** *Finding the shortest splitting cycle on a combinatorial surface is NP-Hard.*

**Proof.** A *grid graph of size* $n$ is a graph induced by a set of $n$ points on the two-dimensional integer grid. We describe a reduction from the Hamilton cycle problem in grid graphs [9].

We describe a two-step reduction. Let $H$ be a grid graph of size $n$. To begin the first reduction, we overlay $n$ $4 \times 4$ square grids of width $\epsilon < 1/4n$, one centered on each vertex of $H$. In each small grid, we color the square in the second row and second column *red* and the square in the third row and third column *blue* (we fix the origin at the upper left corner). We now easily observe that the following question is NP-complete: *Does the modified grid contain a cycle of length at most $n + 1/2$ that separates the red squares from the blue squares?* Any Hamilton cycle for $H$ can be modified to produce a separating cycle of length at most $n + 1/2$ by locally modifying the Hamilton cycle within each small grid, as shown in Figure 1, top. Conversely, any separating cycle must pass through the center points of all $n$ small grids, which implies that any separating cycle of length at most $n + 1/2$ must contain $n$ grid edges that comprise a Hamilton cycle for $H$.

In the second reduction, we reduce the problem to finding a minimum-length splitting cycle. We isometrically embed the modified grid on a sphere, which we call *Earth*. We remove the red and blue squares to create $2n$ punctures, which we attach to two new punctured spheres, called *heaven* and *hell*. We attach the $n$ punctures in heaven to the $n$ blue punctures on Earth; similarly, we attach the $n$ punctures in hell to the $n$ red punctures on Earth. We append edges of length $2n$ to the resulting surface so that each face of the final embedded graph is a disk. The resulting combinatorial surface $\mathcal{M}(H)$ has genus $2n - 2$ and complexity $O(n)$, and it can clearly be constructed in

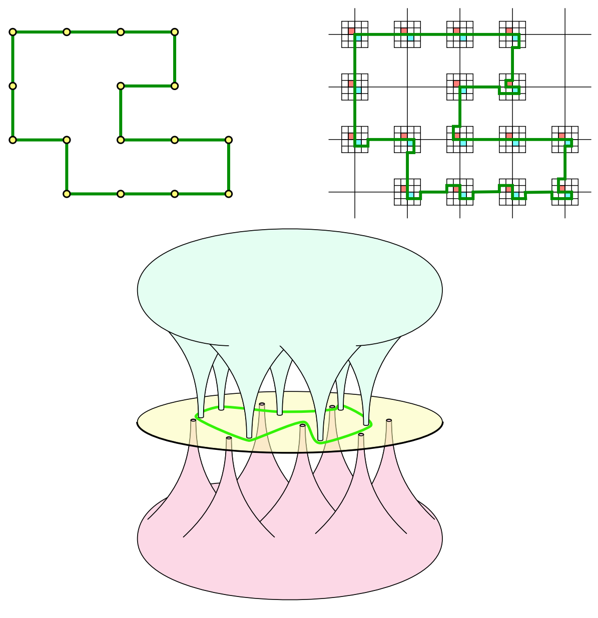


Figure 1: Top left: A Hamilton cycle of length $n$. Top right: The corresponding red/blue separating cycle (not to scale). Bottom: Separating heaven from hell (not to scale); the central disk is a small portion of Earth.

polynomial time. See Figure 1(c).

If the shortest cycle $\gamma$ that splits $\mathcal{M}(H)$ has length less than $n + 1/2$, then it must lie entirely on Earth. Moreover, $\gamma$ must separate the blue punctures from the red punctures; otherwise, $\mathcal{M}(H) \setminus \gamma$ would be connected by a path through heaven or through hell. Thus, $\gamma$ is precisely the shortest cycle that separates the red and blue squares in our intermediate problem. Testing whether $\gamma$ has length less than $n + 1/2$ is thus NP-complete from the first reduction. □

## 4 $O(g)$ Crossings with Any Shortest Path

**Proposition 2** *Let $P$ be a set of pairwise interior-disjoint shortest paths on a cross-metric surface $\mathcal{M}$. Some shortest splitting cycle crosses each path in $P$ at most $O(g)$ times.*

**Proof.** For any two points $x$ and $y$ on a cycle $\alpha$, we let $\alpha[x, y]$ denote the path from $x$ to $y$ along $\alpha$, taking into account the orientation of $\alpha$. For a path or a dual edge $\alpha$, the same notation is used for the unique simple path between $x$ and $y$ on $\alpha$.

Let $\gamma$ be a shortest splitting cycle with the minimum number of crossings with paths in $P$. We can assume that $\gamma$ does not pass through the endpoints of any path in $P$. Consider any path $p$ in $P$ that intersects $\gamma$.

The intersection points $\gamma \cap p$ partition $\gamma$ into *arcs*. These arcs may intersect other paths in $P$. Let $\mathcal{M}/p$ be the quotient surface obtained by contracting $p$ to a point $p/p$. Each arc corresponds to a loop in $\mathcal{M}/p$

with basepoint $p/p$. We say that two such arcs are *homotopic rel* $p$ or *relatively homotopic* if the corresponding loops in $\mathcal{M}/p$ are homotopic.

For any two consecutive intersection points $x$ and $y$ along $\gamma$, the arc $\gamma[x, y]$ cannot be homotopic to $p[x, y]$, since otherwise we can obtain a no longer splitting cycle $\gamma[y, x] \cdot p[x, y]$ that has fewer crossings with the paths in $P$. It follows that none of the arcs of $\gamma$ are contractible rel $p$. Since the arcs are disjoint except at their common endpoints, they correspond, in $\mathcal{M}/p$, to a set of simple, pairwise disjoint loops (except at their common basepoint) that are non-contractible and pairwise non-homotopic. Under these assumptions, the number of loops can be shown to be at most $12g - 6$. Hence there are at most $12g - 6$ relative homotopy classes of arcs.

We can partition the arcs into four types—LL, RR, LR, and RL—according to whether the arcs start on the left or right side of $p$, and whether they end on the left or right side of $p$. To complete the proof, we argue that there is at most one arc of each type in each relative homotopy class. It suffices to consider only types LL and RL; the other two cases follow from symmetric arguments.

Suppose for purposes of contradiction that there are two LL-arcs $u = \gamma[a, z]$ and $w = \gamma[c, x]$ that are homotopic rel $p$. Since the arcs are simple, pairwise disjoint, and homotopic, we may assume without loss of generality that the intersection points appear along $p$ in the order $a, c, x, z$, and the cycle $u \cdot p[z, x] \cdot \bar{w} \cdot p[c, a]$ bounds a disk. Without loss of generality, we assume that no other arc homotopic rel $p$ intersects this disk. Since $\gamma$ is separating, there must be exactly one arc $v = \gamma[y, b]$ between $u$ and $w$ that is relatively homotopic to $\bar{u}$ and $\bar{w}$.

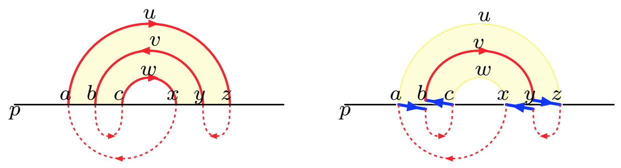


Figure 2: The exchange argument for LL arcs.

Without loss of generality, suppose the path $\gamma[x, a]$ does not contain any of the arcs $u$, $v$, or $w$. Consider the cycle

$$\gamma' = p[a, b] \cdot \gamma[b, c] \cdot p[c, b] \cdot \bar{\gamma}[b, y] \cdot p[y, z] \cdot \gamma[z, y] \cdot p[y, x] \cdot \gamma[x, a]$$

obtained by removing $u$ and $w$ from $\gamma$, reversing $v$, and connecting the remaining pieces of $\gamma$ with subpaths of $p$; see Figure 2. This cycle crosses $p$ fewer times than $\gamma$, and crosses any other path in $p$ no more than $\gamma$. We can prove that $\gamma'$ is simple, separating, and non-contractible. Because $p$ is a shortest path, $u$ cannot be shorter than $p[a, z]$, which implies that $\gamma'$ is no longer than $\gamma$, contradicting the fact that $\gamma$ is a shortest splitting cycle with the minimal number of crossings with paths in $P$. We conclude that any two LL-arcs must be in different relative homotopy classes.

The case of RL-arcs can be treated with a similar, though more complicated, analysis and exchange argument. □

## 5 Algorithm

We will prove the following result.

**Theorem 3** *Let $\mathcal{M}$ be an orientable cross-metric surface without boundary; let $g$ be its genus and $n$ be its complexity. We can compute a shortest splitting cycle in $\mathcal{M}$ in $g^{O(g)} n \log n$ time.*

The algorithm proceeds in several stages, described in the following subsections. First, we compute the shortest system of loops from some arbitrary basepoint. Next, we enumerate all the plausible sequences of intersections of the splitting cycle with the loops in the system of loops. Note that a sequence of intersections determine the homotopy type for the splitting cycle. We discard any sequence that does not yield a valid splitting cycle. Finally, for each remaining sequence, we compute the shortest cycle with the same sequence of intersections. Out of all cycles constructed this way, the shortest one is the correct result.

### 5.1 Greedy Loops

Let $v$ be any point of $\mathcal{M}$ in the interior of a face of $G^*(\mathcal{M})$. Let $\alpha_1, \ldots, \alpha_{2g}$ be the shortest system of loops of $\mathcal{M}$ with basepoint $v$; this system of loops can be computed in $O(n \log n + gn)$ time using a greedy algorithm of Erickson and Whittlesey [6].

Two key properties of this system of loops are that each loop $\alpha_i$ is as short as possible in its homotopy class, and is composed of two shortest paths $\beta_i$ and $\beta'_i$ in the primal graph $G(\mathcal{M})$. However, in general, these two paths meet at a point in the interior of some edge. We can handle this easily by a local modification of the primal or dual graph.

### 5.2 Enumeration of Plausible Sequences via Labeled Triangulations

Lemma 2 tells us that some shortest splitting cycle $\gamma$ crosses each path $\beta_i$ or $\beta'_i$ at most $O(g)$ times, and thus crosses each loop $\alpha_i$ at most $O(g)$ times. Our algorithm therefore enumerates all sequences of intersections where each $\alpha_i$ is crossed at most $O(g)$ times. Since we are using the shortest system of loops, we may assume that the shortest splitting cycle does not cross any $\alpha_i$ consecutively in opposite directions.

Cut $\mathcal{M}$ along the loops $\alpha_i$ to obtain a polygonal schema. This operation also cuts the unknown cycle $\gamma$ into many segments that go across the schema. Since $\gamma$ is simple, no two of these segments cross. Without loss of generality, we can assume that $\gamma$ does not pass through the basepoint $v$.

The segments of $\gamma$ can be grouped into subsets according to which pair of edges they meet on the polygonal schema. We dualize the polygonal schema, replacing each edge with a vertex and connecting vertices that correspond to consecutive edges; now each subset of segments corresponds to an edge between two vertices of the dual $4g$-gon. Since no two segments cross, these edges cannot cross; in particular, all the edges belong to some triangulation of the dual polygon.

Thus the candidate sequences of intersections of a shortest splitting cycle are described by *labeled triangulations*, each of which consists of a triangulation of the dual polygon, in which every edge is labeled with an integer between 0 and $O(g)$. Intuitively, the label of an edge in the triangulation represents the number of times that the cycle runs along that edge. There are $C_{4g-2} = O(4^{4g})$ possible triangulations, where $C_n$ is the $n$th Calatan number, which we can enumerate in $O(g)$ time each. There are $g^{O(g)}$ ways to label each triangulation, which we can enumerate in constant amortized time per labeling. We thus obtain a total of $g^{O(g)}$ potential labeled triangulations for $\gamma$.

## 5.3 Discarding Irrelevant Labeled Triangulations

Most of the labeled triangulations do not correspond to a splitting cycle, or to any cycle for that matter. We now explain how to throw away these possibilities. Namely, given a candidate labeled triangulation $T$, we want to decide if it (1) corresponds to a set of cycles, (2) is actually a single cycle, (3) is separating and (4) is non-contractible. (1) is satisfied if and only if, for each $i$, $\alpha_i$ and $\bar{\alpha}_i$ are crossed the same number of times. If this condition is satisfied, we can build a (set of) simple (pairwise disjoint) cycle(s) on $\mathcal{M}$ representing $T$, of complexity $O(g^2 n)$; for example, the segments can run along the boundary of the polygonal schema. If the second condition is satisfied, we can check conditions (3) and (4) using (simplified versions of) the algorithms by Erickson and Har-Peled [5]. This takes $O(g^2 n \log n)$ time. Actually, this whole step can be done more efficiently in $O(g^2)$ time.

## 5.4 Shortest Cycle for each Sequence of Intersections

In the last step, for each of the $g^{O(g)}$ non-discarded labeled triangulation, we compute the shortest cycle with the same sequence of intersections as defined by the labeled triangulation and keep the shortest one.

For this, we build a cylindrical surface by gluing copies of the polygonal schema. There is one copy per intersection in the sequence. If the $i$th intersection is $\alpha_j$, then the $i$th copy is glued to the $i+1$th copy along $\alpha_j$. (Each loop appears twice on the boundary of the polygonal schema and we must take orientation into account to make the proper gluing. Also, by construction, copies $i-1$ and $i+1$ are glued on different edges on the $i$th copy.) The last copy is glued to the first copy. The resulting cylinder is made of $O(g^2)$ copies of the polygonal schema, each of complexity $O(gn)$. By [7] (see also [2]) we can compute a shortest cycle homotopic to the boundaries of this cylinder in time $O(g^3 n \log n)$. Such a cycle, when projected back onto the surface $\mathcal{M}$, is a shortest cycle with the same prescribed sequence of intersections with the greedy system of loops. The total time spent is $g^{O(g)} n \log n$.

Finally, the output cycle may contain self-intersections. However, we are able to remove these intersections in the same amount of time. This concludes the proof of Theorem 3.

**Acknowledgments.** Support of a travel grant from UIUC/CNRS/INRIA is aknowledged. The authors also thank Martin Kutz for simplifying Section 5.4.

## References

[1] S. Cabello and B. Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. In *Proc. 13th Annu. European Sympos. Algorithms*, volume 3669 of *LNCS*, pages 131–142. Springer-Verlag, 2005.

[2] É. Colin de Verdière and J. Erickson. Tightening non-simple paths and cycles on surfaces. In *Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algorithms*, page to appear, 2006.

[3] É. Colin de Verdière and F. Lazarus. Optimal pants decompositions and shortest homotopic cycles on an orientable surface. In *Proc. 11th Sympos. Graph Drawing*, volume 2912 of *LNCS*, pages 478–490. Springer-Verlag, 2003.

[4] É. Colin de Verdière and F. Lazarus. Optimal systems of loops on an orientable surface. *Discrete Comput. Geom.*, 33(3):507–534, 2005.

[5] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. *Discrete Comput. Geom.*, 31(1):37–59, 2004.

[6] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. In *Proc. 16th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1038–1046, 2005.

[7] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[8] A. Hatcher. *Algebraic topology.* Cambridge University Press, 2002.

[9] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[10] F. Lazarus, M. Pocchiola, G. Vegter, and A. Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 80–89, 2001.

[11] B. Mohar and C. Thomassen. *Graphs on Surfaces.* Johns Hopkins University Press, 2001.

# Pants Decomposition of the Punctured Plane

Sheung-Hung Poon* Shripad Thite*

**Abstract**

A *pants decomposition* of an orientable surface $\Sigma$ is a collection of simple cycles that partition $\Sigma$ into *pants*, i.e., surfaces of genus zero with three boundary cycles. Given a set $P$ of $n$ points in the plane $\mathbb{E}^2$, we consider the problem of computing a pants decomposition of $\Sigma = \mathbb{E}^2 \setminus P$ of minimum total length. We give a polynomial-time approximation scheme using Mitchell's guillotine rectilinear subdivisions. We give an $O(n^4)$-time algorithm to compute the shortest pants decomposition of $\Sigma$ when the cycles are restricted to be axis-aligned boxes, and an $O(n^2)$-time algorithm when all the points lie on a line; both exact algorithms use dynamic programming with Yao's speedup.

## 1 Introduction

*Surfaces* (2-*manifolds*), such as spheres, cylinders, tori, and more, are commonly encountered topological spaces in applications like computer graphics and geometric modeling. To understand the topology of the surface or to compute various properties of the surface, it is useful to *decompose* the surface into simple parts. Among the possible ways to decompose a given surface, it is desirable to compute an *optimum* decomposition, one that minimizes a metric depending on the application.

A decomposition of an orientable surface $\Sigma$ that has been studied is a *pants decomposition* [3, 2], a collection of disjoint cycles that partition $\Sigma$ into *pants*, where a *pant*[1] is a surface of genus zero with three boundary cycles. Every compact orientable surface—except the sphere, disk, cylinder, and torus—admits a pants decomposition [2].

A natural measure of a pants decomposition to minimize is the total length of its boundary cycles. The *length* of a pants decomposition $\Pi$ of $\Sigma$, denoted by $|\Pi|$, is the sum of the (Euclidean) lengths of all the cycles in $\Pi$. (If a subsegment is traversed more than once, its length is counted with multiplicity.) A *non-crossing pants decomposition* is a pants decomposition that allows any two cycles to touch as long as they do not cross transversely. A *shortest* pants decomposition is a non-crossing pants decomposition of minimum length.

The problem of computing a shortest pants decomposition of an arbitrary surface $\Sigma$ is open. In this paper, we study a variant of the problem where $\Sigma$ is the *punctured plane*, i.e., $\Sigma = \mathbb{E}^2 \setminus P$ where $P$ is a discrete set of $n$ points. Figure 1(i) gives an example pants decomposition of the plane with 6 punctures. Colin de Verdière and Lazarus [2] studied a related problem: given a pants decomposition of an arbitrary surface, they compute a *homotopic* pants decomposition in which each cycle is a shortest cycle in its homotopy class.

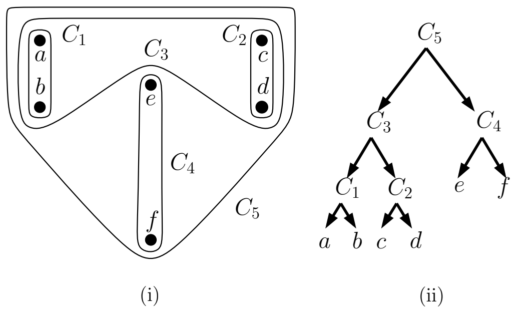


Figure 1: (i) A pants decomposition of the plane punctured by a set of 6 points, and (ii) the corresponding binary tree.

A cycle $C$ on $\Sigma$ is *essential* if it does not bound a disk or an annulus. A pants decomposition, also called a *maximal cut system* [3], is naturally obtained by the following greedy procedure. Let $C$ be an essential cycle. Cut $\Sigma$ along $C$ to get a surface $\Sigma'$ with two additional boundary cycles $C_1$ and $C_2$ corresponding to the two sides of the cut. The cycles $C_1$ and $C_2$, together with the set of cycles obtained by recursing on $\Sigma'$, is a pants decomposition $\Pi$ of $\Sigma$. The resulting cycle structure can be modeled as a binary tree with $n$ leaves (Figure 1(ii)). Each cycle $C$ of $\Pi$ is essential because it encloses two other cycles, say $C_1$ and $C_2$; we write $C_1 \prec C$ and $C_2 \prec C$ to indicate that $C$ encloses $C_1$ and $C_2$. We say that two cycles $C_1$ and $C_2$ are *independent* if neither $C_1 \prec C_2$ nor $C_2 \prec C_1$. The final result is a pants decomposition of a bounded subset of the plane together with a single unbounded

*Dept. of Math and Computer Sci., Technische Universiteit Eindhoven, The Netherlands; {spoon,sthite}@win.tue.nl.
S.-H.P. and S.T. were supported by the Netherlands' Organisation for Scientific Research (NWO) under project numbers 612.065.307 and 639.023.301 respectively.

[1]We call a surface of genus zero with three boundary components a *pant* instead of a *pair of pants* [2]. We refer to two pants instead of two pairs of pants; the latter phrase can be misunderstood to mean four such surfaces.

component.

In this paper, we give a simple algorithm with approximation ratio $O(\log n)$, and a polynomial-time approximation scheme (PTAS) using Mitchell's guillotine rectilinear subdivisions. We compute the shortest pants decomposition in $O(n^4)$ time when the cycles are restricted to be axis-aligned boxes, and in $O(n^2)$ time when all the points lie on a line; both exact algorithms use dynamic programming with Yao's speedup, and are faster by a linear factor than the 'naïve' dynamic programming formulations.

## 2 A simple approximation algorithm

If $\Pi^*$ is a shortest pants decomposition of the punctured plane $\Sigma = \mathbb{E}^2 \setminus P$, then every cycle in $\Pi$ is a simple polygon whose vertices belong to $P$. We argue next that $\Pi^*$ contains a traveling salesperson (TSP) tour of the points. Choose any vertex on the outermost cycle as the start of the tour. Traverse the outermost cycle counterclockwise. The first time we visit a vertex $u$ that also belongs to an inner cycle (one of the two legs of the pant) that has not been traversed yet, we recursively construct a tour beginning and ending at $u$ that traverses the unvisited vertices on or in the interior of this inner cycle. After the recursion, we continue along the outermost cycle, repeating the recursive traversal of the second leg of the pant, until we reach our original starting point. Hence, $\Pi^*$ must be at least as long as a shortest Euclidean TSP tour $T^*$ of the points. Hence, $|\Pi^*| \geq |T^*|$.

We convert a TSP tour $T$ of the points in $P$ to a pants decomposition $\Pi$ as follows. Initially, $\Pi$ is the empty set. Order the points from 0 through $n-1$ in the order along the tour $T$. Let $C(i,j)$ denote the polygon with vertices $i$, $i+1$, $i+2$, …, $j-2$, $j-1$, $j$, $j-1$, $j-2$, …, $i+2$, $i+1$, $i$ in order, where the indices are taken modulo $n$. Imagine cycles of zero length around each point. Repeatedly introduce a new cycle into $\Pi'$ that is obtained by merging the two cycles adjacent along the tour enclosing the fewest number of points. Each cycle $C(i,j)$ is obtained by merging two cycles $C(i,k)$ and $C(k+1,j)$ by doubling the edge between vertices $k$ and $k+1$. We ensure that each edge in the tour $T$ appears exactly twice in at most $\lceil \log n \rceil$ cycles in the pants decomposition $\Pi$. Hence, $|\Pi| \leq 2\lceil \log n \rceil |T|$.

It is well-known [7] how to obtain a $3/2$-approximate shortest Euclidean TSP of the point set using Christofides' algorithm in $O(n^3)$ time; the minimum spanning tree of the $n$ points can be used to obtain a 2-approximation in $O(n \log n)$ time. An approximate TSP tour obtained by either of these algorithms gives us a non-crossing pants decomposition of length $O(\log n)$ times the optimum.

## 3 PTAS

Let $\varepsilon > 0$ be an arbitrary constant. To construct in polynomial time a $(1+\varepsilon)$-approximation to the shortest non-crossing pants decomposition, we modify the PTAS for Euclidean TSP tour due to Mitchell [5, 6]. Our algorithm is more complicated because a pants decomposition consists of $\Theta(n)$ cycles instead of just one cycle as in a TSP tour. The PTAS is a dynamic programming algorithm where each subproblem is a rectangular region of the plane and two adjacent subproblems interact only through $O(1)$ grid points or *portals*.

Let $m \geq 2$ be an integer and let $M = m(m-1)$. Let $B$ denote the axis-aligned bounding box of the point set $P$. Imagine a shortest pants decomposition $\Pi^*$. Mitchell [6] has shown that there exists a *favorable cut*, i.e., a horizontal or vertical line $l$, which partitions $B$ into two smaller boxes that can be recursively subdivided using favorable cuts. The recursion stops when a box is empty of points of $P$. Just like Mitchell, we introduce a segment of $l$, called a *bridge*, and $O(M)$ grid points on $l$. Mitchell has shown that the total length of the additional subsegments is at most $\frac{\sqrt{2}}{m}$ times the length of $\Pi^*$.

Let $R$, a rectangle, be the boundary of an arbitrary box $Q$ during the recursive subdivision. Intuitively, we can "bend" the cycles of $\Pi^*$ to make each cycle that crosses $R$ transversely do so only at one of the portals (grid points) and possibly use subsegments of the bridges on the four sides of $R$, without increasing the length of the pants decomposition by too much.

To construct a short pants decomposition $\Pi$, our PTAS solves subproblems of the following form. We are given a rectangle $R$ whose sides are defined by two horizontal and two vertical favorable cuts. We are given two integers $n_i$ and $n_t$, both in the range from 0 through $n-1$, of the number of cycles of $\Pi$ that are inside $R$ and that intersect $R$ transversely, respectively. Each of the $n_i$ cycles inside $R$ intersects $R$ tangentially and an even number of times, and each of the $n_t$ cycles intersects $R$ transversely and an odd number of times. Let $n_R = n_i + n_t$. We are given the pattern in which the $n_R$ cycles intersect $R$ at the $O(M)$ grid points on the sides of $R$. There are $O(n^{O(M)})$ possible ways for the $n_R$ cycles to intersect the sides of $R$.

It remains to account for the fact that cycles in $\Pi$ that intersect $R$ tangentially can traverse subsegments of the cuts bounding $R$. We observe that every point in the plane lies on at most two independent cycles of $\Pi$. Let $p$ be an arbitrary point in the plane. Let $C_p$ denote the subset of cycles in $\Pi$ that pass through $p$. If $|C_p| > 2$, then there exist three cycles $C_1$, $C_2$, and $C_3$ in $C_p$ such that both $C_1$ and $C_2$ are inside $C_3$. Let $C$ be an outermost (minimum depth) cycle that traverses a subsegment $ab$ of some cut. Each subsegment is shared by at most two independent cycles.

Therefore, we count the length of $ab$ at most twice when counting the total length of all subsegments of cuts traversed.

The dynamic programming algorithm proceeds as follows. For a rectangle $R$ intersected by $n_R = n_i + n_t$ cycles, we try each of the $O(n)$ favorable cuts that partition $R$ into two smaller rectangles, $A$ and $B$. We try each of the $O(n_R^{O(M)})$ possible ways that the $n_R$ cycles can intersect the cut transversely, making sure that the pattern in which the cycles intersects the cut is consistent with the pattern in which they intersect $R$. Some of the $n_i$ cycles that belong inside $R$ may belong inside $A$ and some others inside $B$; we try the $O(n_i)$ possible ways to allocate a subset of the $n_i$ cycles to $A$ and the remaining to $B$. We optimize over the $O(n)$ cuts and $O(n^{O(M)})$ intersection patterns to solve the subproblem $R$ optimally. In the base case, if $R$ has no points of $P$ in its interior, then the subproblem has only $O(M)$ size, which is a constant, and is solved by brute force. Since there are $O(n^{O(M)})$ different subproblems and each subproblem takes $O(n^{O(M)})$ time, the total running time is $O(n^{O(M)})$.

The length of the pants decomposition $\Pi$ obtained by the dynamic programming algorithm is $O\left(1 + \frac{2\sqrt{2}}{m}\right)$ times that of a shortest pants decomposition. To obtain the desired approximation factor, we choose $m \geq 2\sqrt{2}/\varepsilon$.

To reiterate, the major differences between our PTAS and that of Mitchell [5] are the following. (i) Each of our $n-1$ cycles crosses transversely the boundary of a rectangular subproblem in one of $O(M)$ different ways. Therefore, we have $O(n^{O(M)})$ times as many subproblems to solve as in the TSP. (ii) Each of the $O(M)$ grid points on the boundary of a rectangular subproblem may lie on any of the $n-1$ cycles in a pants decomposition. Therefore, the additional information associated with each subproblem is more than of constant size; the amount of information associated with each subproblem is $O(n^{O(M)})$. However, the total running time is still polynomial in $n$.

## 4 Points on a line

Let $P$ be a set of points on a line, without loss of generality on the $x$-axis. Order the $n$ points from left to right. Let $x_i$ denote the $x$-coordinate of the $i$th point. For every $1 \leq i \leq j \leq n$, let $(i, j)$ denote the $j-i+1$ consecutive points numbered from $i$ through $j$.

We prove next that a shortest non-crossing pants decomposition of $\Sigma = \mathbb{E}^2 \setminus P$ must consist of convex cycles only. Hence, if $\Pi^*$ is a shortest non-crossing pants decomposition of $\Sigma$, then there exists a $k$ in the range $1 \leq k < n$ such that $\Pi^*$ consists of a shortest pants decomposition of $(i, k)$ and a shortest pants decomposition of $(k+1, n)$ together with an outermost cycle of length $2(x_n - x_1)$ enclosing all $n$ points.

Suppose to the contrary that there is a cycle $C$ in $\Pi^*$ that contains a non-contiguous subset of points in $P$. Without loss of generality, we assume that $C$ is a minimal one in the sense that both its legs, $C_1$ and $C_2$, contains contiguous subsets of points. Without loss of generality, assume $C_1$ is to the left of $C_2$; thus, $C_1$ is the *left leg* and $C_2$ is the *right leg* of $C$.

Let $X \subseteq P$ be the set of all points between $C_1$ and $C_2$. Let $x \in X$ be arbitrary. Let $\mathcal{D}_x = \{D_1, D_2, D_3, \ldots, D_i, \ldots\}$ be the set of cycles in $\Pi^*$ containing $x$ such that $D_{i+1}$ contains $D_i$. Let $i$ be the smallest index such that $D_i$ contains some point of $P \setminus X$. There are two cases to consider: (i) $D_i$ contains $C$, (ii) $D_i$ does not contain $C$.

If $D_i$ contains $C$, then we construct another cycle $E$ enclosing $C_1$ and $D_{i-1}$ making $E$ the left leg of $C$ (instead of $C_1$). Delete $D_i$.

Otherwise, if $D_i$ does not contain $C$, then either (a) $D_i$ contains points only to the left of $C_2$ or (b) $D_i$ contains points only to the right of $C_1$. In the former case, $D_{i-1}$ is the *right leg* of $D_i$. We swap $C_1$ and $D_{i-1}$ so that $C_1$ is the new right leg of $D_i$ and $D_{i-1}$ is the new left leg of $C$. In the latter case, $D_{i-1}$ is the *left leg* of $D_i$. We swap $C_2$ and $D_{i-1}$ so that $C_2$ is the new left leg of $D_i$ and $D_{i-1}$ is the new right leg of $C$.

In either case, we obtain a pants decomposition with total length smaller than $\Pi^*$, which is a contradiction.

Let $c(i, j)$ denote the cost of a shortest pants decomposition of $(i, j)$. We have just proved that $c(i, j)$ satisfies the following recurrence for every $1 \leq i \leq j \leq n$:

$$c(i, j) = 2(x_j - x_i) + \min_{i \leq k < j} (c(i, k) + c(k+1, j)) \quad (1)$$

where $c(i, i) = 0$. A shortest pants decomposition of $(i, j)$ can be computed by choosing the appropriate value of $k$ in the range $i \leq k < j$, computing the optimum pants decompositions of $(i, k)$ and $(k+1, j)$, and introducing a non-crossing cycle of length $2(x_j - x_i)$ enclosing the points $(i, j)$. The straightforward dynamic programming algorithm computes $c(1, n)$ in $O(n^3)$ time.

We show how the running time of the dynamic programming algorithm can be improved by a linear factor using Yao's speedup [9]. Let $w(i, j) = x_j - x_i$. The function $w()$ is *monotone*, i.e., $w(i, j) \leq w(k, l)$ whenever $(i, j) \subseteq (k, l)$, and satisfies the following *concave quadrangle inequality* [9]:

$$\forall\, i \leq i' \leq j \leq j' : w(i, j) + w(i', j') \leq w(i', j) + w(i, j')$$

In fact, the above equation is satisfied with equality because $w(i, j) + w(i', j') = (x_j - x_i) + (x'_j - x'_i) = w(i', j) + w(i, j')$. Let $c_k(i, j)$ denote $w(i, j) + c(i, k) + c(k+1, j)$. Let $K(i, j)$ denote the maximum $k$ for which $c(i, j) = c_k(i, j)$. The following claims are almost identical to those in the context of optimum binary search trees proved by Mehlhorn [4, 8]:

1. The function $c(i,j)$ also satisfies the concave quadrangle inequality, i.e.,
$$\forall\, i \le i' \le j \le j' : c(i,j) + c(i',j') \le c(i',j) + c(i,j')$$
2. $K(i,j-1) \le K(i,j) \le K(i+1,j)$

We compute $c(i,j)$ by diagonals, in order of increasing value of $j-i$. For each fixed difference $d$, we compute $c(i,j)$ where $j = i+d$; we compute $c_k(i,j)$ for $k$ in the range $K(i,j-1) \le k \le K(i+1,j)$. The cost of computing all entries on the $d$th diagonal is

$$\begin{aligned}&\sum_{i=1}^{n-d} K(i+1,j) - K(i,j-1) + 1\\ &= K(n-d+1,n+1) - K(1,d) + n - d\\ &\le (n+1) - 1 + n - d\\ &< 2n\end{aligned}$$

Since $d$ ranges from 0 through $n-1$, the total running time is $O(n^2)$.

## 5 Box decomposition

A *box decomposition* of $\Sigma$ is a pants decomposition $\Pi$ in which each cycle in $\Pi$ is an axis-aligned rectangle. Observe that any two axis-aligned rectangles can be separated from each other by either a horizontal or a vertical line.

Let $x_1$ through $x_n$ denote the $x$-coordinates of the $n$ points in increasing order, and let $y_1$ through $y_n$ denote the $y$-coordinates of the $n$ points in increasing order. Let $h(i,j) = 2(x_j - x_i)$ and let $v(i,j) = 2(y_j - y_i)$. Let $w(i_1,i_2,j_1,j_2) = h(i_1,i_2) + v(j_1,j_2)$; then, $w(i_1,i_2,j_1,j_2)$ is the perimeter of the axis-aligned box whose sides have $x$-coordinates $x_{i_1}$ and $x_{i_2}$ and $y$-coordinates $y_{j_1}$ and $y_{j_2}$. The function $w()$ is *monotone* and satisfies the *concave quadrangle inequality.*

The rest of the proof is similar to the case for points on a line. Let $c(i_1,i_2,j_1,j_2)$ denote the cost of a shortest pants decomposition of the points in the box $[x_{i_1}, x_{i_2}] \times [y_{j_1}, y_{j_2}]$. The cost $c(i_1,i_2,j_1,j_2)$ obeys a recurrence that is a two-dimensional generalization of Equation 1. Similar to the dynamic programming algorithm for points on a line, we compute $c(i_1,i_2,j_1,j_2)$ by diagonals, in order of increasing value of $\max\{i_2 - i_1, j_2 - j - 1\}$. For each pair of differences $d_1$ and $d_2$, we compute $c(i_1,i_2,j_1,j_2)$ where $i_2 = i_1 + d_1$ and $j_2 = j_1 + d_2$. The cost of computing all entries on the diagonals defined by $(d_1,d_2)$ is $O(n^2)$; since there are $O(n^2)$ such pairs $(d_1,d_2)$, the total running time is $O(n^4)$.

## 6 Work in progress

We mention some very interesting open questions that we are currently investigating.

Is it NP-hard to determine, for an arbitrary $L$, whether there exists a non-crossing pants decomposition of the punctured plane of length at most $L$? Is there a simple algorithm to compute an $O(1)$-approximate shortest pants decomposition?

Are the cycles in a shortest (non-crossing) pants decomposition always convex? If not, how much longer than optimum is a convex pants decomposition?

How efficiently can we compute a shortest pants decomposition of the plane with different types of punctures, e.g., rectangular holes instead of points?

How efficiently can we compute a shortest pants decomposition of other 2-manifolds, such as the torus minus a set of points?

## Acknowledgments

We thank members of the Algoritmiek group at TU/e, especially Mark de Berg, for fruitful discussions. Shripad Thite thanks the organizers and participants of the Meeting on Optimization Problems in Computational Topology at ENS, Paris, in November 2005, especially Éric Colin de Verdière and Jeff Erickson.

## References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. J. ACM, 45(5):753–782, 1998.

[2] É. Colin de Verdière, F. Lazarus. Optimal Pants Decompositions and Shortest Homotopic Cycles on an Orientable Surface. Proc. Graph Drawing, pp. 478–490, 2003. Preliminary version at EuroCG'03.

[3] A. Hatcher. Pants Decompositions of Surfaces. arXiv:math.GT/9906084; `http://arxiv.org/abs/math.GT/9906084`

[4] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* EATCS Monographs on Theoret. Comput. Sci., Springer-Verlag, 1984.

[5] J. S. B. Mitchell. Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, $k$-MST, and Related Problems. SIAM J. Computing, 28(4):1298–1309, 1999.

[6] J. S. B. Mitchell. Approximation Algorithms for Geometric Optimization Problems. Proc. Canadian Conf. Comput. Geom., pp. 229–232, 1997.

[7] C. H. Papadimitriou, K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

[8] S. Thite. Optimum Binary Search Trees on the Hierarchical Memory Model. M.S. thesis, Computer Sci., U. Illinois at Urbana-Champaign, CSL Tech. Rep. UILU-ENG-00-2215 ACT-142, 2000.

[9] F. F. Yao. Speed-up in Dynamic Programming. SIAM J. Algebraic Discrete Methods, 3(4):532–540, 1982.

# Computing the Fréchet Distance Between Simple Polygons

Kevin Buchin*[†], Maike Buchin*[†], Carola Wenk[‡]

## Abstract

We present the first polynomial-time algorithm for computing the Fréchet distance for a non-trivial class of surfaces: simple polygons. For this, we show that it suffices to consider homeomorphisms that map an arbitrary triangulation of one polygon to the other polygon such that diagonals of the triangulation are mapped to shortest paths in the other polygon.

## 1 Introduction

The Fréchet distance is a distance measure used in shape matching. It is defined for continuous shapes such as curves and surfaces using reparametrizations of the shapes.

The Fréchet distance between polygonal curves can be computed in polynomial time [2], however computing the Fréchet distance distance for (two-dimensional) surfaces is NP-hard [5]. Except for the NP-hardness very little is known so far about the Fréchet distance of surfaces. It is known to be semi-computable [1], but it is unknown whether it is computable, and there are no approximation algorithms.

We address this problem by considering a restricted but important class of surfaces, simple polygons, and show that their Fréchet distance can be computed in polynomial time. This is the first polynomial-time algorithm for computing the Fréchet distance for a non-trivial class of surfaces.

The rest of this abstract is organized as follows: First we introduce in Section 2 notations and preliminary lemmas. Then we show in Section 3 that it suffices to look at a small well-behaved class of homeomorphisms. We use this to develop a polynomial time algorithm for deciding the Fréchet distance in Section 4 which we extend to a computation algorithm by searching over a set of critical values.

Due to space restrictions we omit detailed proofs in this extended abstract but provide the main ideas.

*Freie Universität Berlin, Institute of Computer Science, Takustr. 9, 14195 Berlin, Germany

[†]This research was supported by the Deutsche Forschungsgemeinschaft within the European graduate program 'Combinatorics, Geometry, and Computation' (No. GRK 588/2).

[‡]University of Texas at San Antonio, Department of Computer Science, 6900 N. Loop 1604 West, San Antonio, TX 78249-0667, USA

## 2 Preliminaries

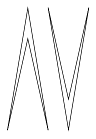
### Simple Polygons

Let $P$ and $Q$ be two simple polygons in the plane with $m$ and $n$ vertices, respectively. A simple polygon is the area enclosed by a non-selfintersecting closed polygonal curve in the plane. The two polygons may lie in two different planes. We assume as underlying parametrizations the identity maps $f : P \to P$ and $g : Q \to Q$. Their Fréchet distance is:

$$\delta_F(P,Q) = \inf_{\sigma:P\to Q} \max_{t\in P} ||t - \sigma(t)|| \,,$$

where $\sigma$ ranges over all orientation-preserving homeomorphisms and $||.||$ is the Euclidean norm[1]. In the remainder we will only consider orientation-preserving homeomorphisms and might refer only to $\sigma$ or to a homeomorphism when the meaning is clear from the context.

The first question that comes to mind is: Is the Fréchet distance of polygons different from the Fréchet distance of their boundary curves?



**Observation 1** *The Fréchet distance of two polygons may be arbitrarily larger than the Fréchet distance of their boundary curves.*

This observation can be proved by showing that it holds for the two polygons on the right.

We will compute the Fréchet distance between simple polygons using shortest paths and for this use an important concept which was introduced by Guibas et al. [6]: *hourglasses*. If $s_1$ and $s_2$ are two segments in a simple polygon, the hourglass of $s_1$ and $s_2$ represents all shortest paths between any point on $s_1$ and any point on $s_2$.

### Simplifying a Curve

Given a curve $f$ and a line segment $s$, Lemma 1 shows that simplifying $f$ by replacing a part of it with a line segment does not increase the Fréchet distance to $s$.

**Lemma 1** *Let $f : [0,1] \to \mathbb{R}^d$ be a curve, let $s : [0,1] \to \mathbb{R}^d$ be a line segment, and let $0 \le t_1 < t_2 \le 1$. Define $f' : [0,1] \to \mathbb{R}^d$ to be equal to $f$ for $t \in [0,t_1] \cup$*

[1]Of course any other metric can be considered as well.

$[t_2, 1]$. *And for* $t \in [t_1, t_2]$ *it equals the line segment from* $f(t_1)$ *to* $f(t_2)$. *Then* $\delta_F(f', s) \leq \delta_F(f, s)$.

For proving this lemma, we show that a homeomorphism $\sigma$ for computing $\delta_F(f, s)$ can be modified to a homeomorphism $\sigma'$ for computing $\delta_F(f', s)$ that does not yield a larger value. For this we use that the maximum distance between two segments is attained at segment end points.

## 3 Shortest Paths Lemma

The next lemma states that it suffices to look at homeomorphisms that map the diagonals of a triangulation of $P$ to shortest paths in $Q$ and are piecewise linear inside triangles. For a triangulation $T$ of $P$ we denote with $E_T$ the set of points lying on all edges of $T$, i.e., all points on all boundary edges and diagonals (and vertices) of $T$.

**Lemma 2** *Given two simple polygons* $P$ *and* $Q$, *a triangulation* $T$ *of* $P$ *and homeomorphism* $\sigma : P \to Q$. *Then there is a map* $\sigma' : P \to Q$ *which fulfills*

(1) $\sigma'$ *is a limit of homeomorphisms from* $P \to Q$

(2) $\sigma'$ *maps the diagonals of the triangulation* $T$ *to shortest paths in* $Q$ *and is piecewise linear inside triangles without introducing interior vertices. Thus* $\max_{t\in P} ||t - \sigma'(t)|| = \max_{t\in E_T} ||t - \sigma'(t)||$.

(3) $\sigma'$ *is "at least as good as* $\sigma$*", i.e.,* $\max_{t\in P} ||t - \sigma'(t)|| \leq \max_{t\in P} ||t - \sigma(t)||$.

**Proof sketch.** Let $\sigma'$ be as follows: $\sigma'$ equals $\sigma$ on the boundary, it maps diagonals of $T$ to the shortest paths between the corresponding boundary points of $Q$, and is extended piecewise linearly inside triangles (without introducing any interior vertices).

Then (2) holds by definition of $\sigma'$ and (1) holds because the shortest paths may be overlapping but noncrossing. We can show (3) by iteratively shooting rays along the edges of the shortest path and simplifying the curve $\sigma(D)$ using Lemma 1.

From this lemma we get the following corollaries:

**Corollary 3** *The Fréchet distance between simple polygons* $P$ *and* $Q$ *equals*

$$\inf_{\sigma':P\to Q} \max_{t\in T} ||t - \sigma'(t)||$$

*where* $T$ *is an arbitrary triangulation of* $P$. $\sigma'$ *ranges over all homeomorphisms from the boundary of* $P$ *to the boundary of* $Q$ *which are extended to* $T$ *by mapping the diagonals of* $T$ *to the shortest paths between the boundary vertices and which are extended piecewise linearly inside the triangles (without introducing interior vertices).*

**Corollary 4** *The Fréchet distance between two simple polygons, of which one polygon is convex, equals the Fréchet distance between their boundary curves.*

The first corollary follows immediately. For the second corollary we triangulate the possibly non-convex polygon and map the diagonals of the triangulation to shortest paths in the convex polygon. Then the shortest paths in the convex polygon are also diagonals and the Fréchet distance between two line segments equals the maximum distance of its endpoints.

## 4 Computing the Fréchet distance

The main result of this section is a polynomial time algorithm for computing the Fréchet distance between simple polygons. But first we have to show some preliminary results and introduce some more notation.

In the following $P$ and $Q$ always denote two simple polygons, $n$ and $m$ the number of vertices of the boundaries of $P$ and $Q$, respectively, and $\varepsilon$ a real value greater 0. $T$ is a triangulation of $P$. The decision problem is to decide whether $\delta_F(P, Q) \leq \varepsilon$.

### Free Space Diagram and Reachability Structure

The *free space diagram* is the data structure developed by Alt and Godau [2] for computing the Fréchet distance between polygonal curves. For $\varepsilon > 0$ and two parametrized curves $f, g : [0, 1] \to \mathbb{R}^d$ it is defined as $\{(s, t) \in [0, 1]^2 \mid ||f(s) - g(t)|| \leq \varepsilon\}$. If $f$ and $g$ are polygonal curves of complexity $n$ and $m$, respectively, then the free space diagram can be represented in a rectangle $[0, n] \times [0, m]$ consisting of $n$ columns and $m$ rows of a total of $mn$ *cells*. The *double free space diagram* is the free space diagram of $f$ concatenated $f$ and $g$ and can be represented in $[0, 2n] \times [0, m]$.

The decision problem for closed polygonal curves is solved by computing the *reachability structure* [2]. It is a partition of the boundary of the double free space diagram into $O(mn)$ intervals, where each intervals is assigned pointers containing information on the reachability in free space. The reachability structure has complexity $O(mn)$ and can be computed in $O(mn \log mn)$ time.

### Feasible Path in the Free Space Diagram

By Lemma 2 it suffices to consider homeomorphisms on the boundary curves of the simple polygons $P$ and $Q$ and extend these by mapping the diagonals of a triangulation of $P$ to the corresponding shortest paths in $Q$. In other words for the decision problem we search for a homeomorphism $\sigma : P \to Q$ fulfilling:

(1) $\sigma$ maps $\partial P$ onto $\partial Q$ and $\max_{t\in\partial P} ||t - \sigma(t)|| \leq \varepsilon$

(2) $\sigma$ maps all diagonals of a triangulation to shortest paths in $Q$, and all diagonals $D$ must fulfill $\max_{t \in D} ||t - \sigma(t)|| \leq \varepsilon$

For condition (1) we use the algorithm and data structure developed by Alt and Godau [2] for closed curves which searches for a monotone path in the double free space diagram. We handle condition (2) as follows: a path in the free space diagram determines the shortest paths that the diagonals are mapped to because it maps the end points of the diagonals. We call these *diagonal placements*. A path in the free space diagram which places all diagonals correctly, i.e., fulfills (2), we call *feasible path*.

## Order of Diagonals in a Triangulation

The edge set of a triangulation $T$ of $P$ consists of edges on the boundary and in the interior of $P$, the latter of which are *diagonals*. We define an order of the diagonals which we will later use for dynamic programming over the diagonals. For a fixed starting point $s$ on the boundary of $P$ we order the diagonals as follows: If we write the diagonals as ordered tuples of their end points $(d_i, d_j)$ with $i < j$, we define $(d_i, d_j) < (d_k, d_l) :\Leftrightarrow (j < l) \vee (j = l \wedge i > k)$.

For two starting points that lie in between the same two diagonal end points the resulting diagonal order is the same. Hence there are at most $n - 2$ different orders of diagonals in total, each characterized by the next diagonal endpoint (in counterclockwise order) to the starting point. We will call areas of the free space diagram that induce the same diagonal order *blocks*. Blocks consists of one or two columns in the diagram that lie between the vertical lines corresponding to two neighboring diagonal end points.

## Combined Reachability Graph

The *combined reachability graph* combines the reachability information in the free space with valid diagonal placements. First we define a *reachability graph* which is the reachability structure represented as a graph: its vertices are the reachable intervals of the reachability structure with an edge between two intervals if they can reach each other. The combined reachability graph is a subgraph of the reachability graph. Its vertices are all vertical interval-vertices of the reachability graph with edges between intervals that can be reached by feasible paths. For a fixed order of diagonals the combined reachability graph can be computed by recursively merging the reachability graphs of the blocks in the order of diagonals. Since the reachability structure contains $O(mn)$ intervals the reachability graph and the combined reachability graph contain $O(mn)$ vertices and $O((mn)^2)$ edges.

We will use the combined reachability graph as follows: When searching for feasible paths starting in block $B_1$ we compute the combined reachability graph for the order of diagonals starting in $B_1$ by merging blocks $B_2$ through $B_l$ (where $l$ is the number of blocks). A feasible path starting in block $B_1$ consists of an edge in the reachability graph of $B_1$ from its lower to its right boundary, an edge in the combined reachability graph between the left boundary of $B_2$ to the right boundary of $B_l$, and an edge in the reachability graph of $B_1$ from its left to its upper boundary.

## Fréchet Distance of a Diagonal and an Hourglass

The following lemma shows how to decide the Fréchet distance between a diagonal and a whole set of shortest paths, namely the hourglass of two segments. With a *shortest path in the hourglass* we always refer to a shortest path between two points on the two segments defining the hourglass.

**Lemma 5** *Let an hourglass and a diagonal be given such that each end segment of the hourglass is contained in one of the $\varepsilon$-disks around the endpoints of the diagonal (and not both in the same disk). If there exists one shortest path in the hourglass with Fréchet distance at most $\varepsilon$ to the diagonal, then all shortest paths in the hourglass have Fréchet distance at most $\varepsilon$ to the diagonal.*

The idea of the proof of this lemma is the following: If $A = a_1, \ldots, a_l$ is a shortest path in the hourglass with Fréchet distance at most $\varepsilon$ to the diagonal and $B = b_1 \ldots, b_k$ another shortest path in the hourglass. Define $B' = b_1, a_1, \ldots, a_l, b_k$. $B'$ can be simplified to $B$ by repeatedly using Lemma 1.

## Fréchet Distances of a Diagonal and many Hourglasses

In Section 4 we need to decide all Fréchet distances between a diagonal and several hourglasses that have a common end segment. This can be done in $O(m)$ time by choosing an arbitrary vertex on each end segment of the hourglasses and using Lemma 5 and Lemma 6.

**Lemma 6** *Given a diagonal, a polygon with $m$ vertices, and a set of $m$ vertices $w_1, \ldots, w_m$ on the boundary of the polygon. Then we can decide all Fréchet distances between the diagonal and the $m$ shortest paths $\pi(w_1, w_i)$ between $w_1$ and $w_i$ for $i = 1, \ldots, m$ in total $O(m)$ time.*

For proving this lemma we use the linear time algorithm for computing the lengths of all shortest paths from one vertex of a simple polygon to all others by Guibas et al. [6]. During the algorithm we store additional information about reachability in free space, which can be updated in constant time while processing each vertex.

## Decision Algorithm

**Algorithm 1** DecideFréchet($P, Q, \varepsilon$)

```
Input: Simple Polygons $P, Q$, $\varepsilon > 0$
Output: Is $\delta_F(P, Q) \leq \varepsilon$?
1  Compute a triangulation of $P$
2  Compute all orders of diagonals in the triangulation of $P$
3  Compute a single free space diagram of the boundary curves
4  Compute the reachability graph for all blocks in the free space diagram
5  forall diagonals in the triangulation do
6      forall placements in the free space do
7          test $\delta_F$(diagonal, shortest path)$\leq \varepsilon$ for a corresponding shortest path
8      end
9  end
10 forall blocks do
11     forall diagonals, in the order given by the block do
12         if combined reachability graph is not yet computed then
13             if previous diagonal nested then
14                 compute the combined reachability graph merged with the combined reachability graph of the nested diagonal
15             end
16             else
17                 compute the combined reachability graph
18             end
19             store the combined reachability graph
20         end
21         if diagonal has a left neighbor then
22             merge the combined reachability graphs
23         end
24     end
25     Query for a feasible path starting at the lower boundary of the block
26 end
27 Answer “yes” if a feasible path has been found, else “no”
```

**Theorem 7** *Algorithm* DecideFréchet($P, Q, \varepsilon$) *decides whether the Fréchet distance between simple polygons $P, Q$ is at most $\varepsilon$. The runtime is $O(nT(mn))$, where $T(N)$ is the time to multiply two $N \times N$ matrices, and $n$ and $m$ are the number of vertices of $P$ and $Q$.*

Note that $T(N) = \Omega(N^2)$ and the currently fastest known matrix multiplication algorithm has a runtime of $T(N) = O(N^{2.376})$ [4]. Due to space restrictions we have to completely omit the proof of this theorem.

## Critical Values for Computation

For computing the Fréchet distance we apply the same technique as for curves [2]: We search a set of critical values using parametric search. In addition to the critical values of the boundary curves we consider critical values for the Fréchet distance between diagonals and shortest paths.

A shortest path is always a polygonal curve where the first and last vertex are arbitrary points on the boundary of $Q$ and all other vertices are vertices of $Q$. The distances between the diagonal end points and the boundary of $Q$ are already contained in the critical values for the boundary curves. Additional critical values can only occur if the Fréchet distance between a diagonal and a shortest path is attained in the interior of the diagonal and the shortest path, i.e., it is attained at a vertex of $Q$. For the parametric search we sort the ”spikes” in the free space diagram. We get $m \cdot n$ such spikes, one for any pair of a diagonal and a vertex of $Q$. In total we get:

**Theorem 8** *The Fréchet distance between two simple polygons can be computed in time $O(nT(mn)\log(mn))$, where $T(N)$ is the time to multiply two $N \times N$ matrices, and $n$ and $m$ are the number of vertices on the boundary of $P$ and $Q$.*

## Acknowledgements

We thank Boris Aronov for suggesting the problem, and we thank Peter Brass and Helmut Alt for fruitful discussions.

## References

[1] H. Alt and M. Buchin. Semi-computability of the Fréchet distance between surfaces. In *Proc. 21st Europ. Workshop on Comp. Geom.*, pages 45–48, 2005.

[2] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[3] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[5] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions.* PhD thesis, Freie Universität Berlin, Germany, 1998.

[6] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proc. 2nd Ann. Symp. on Comput. Geom.*, pages 1–13, New York, NY, USA, 1986. ACM Press.

# Probabilistic matching of sets of polygonal curves*

Helmut Alt† Ludmila Scharf † Sven Scholz †

## 1 Introduction

Analysis and comparison of geometric shapes are of importance in various application areas within computer science, e.g., pattern recognition and computer vision. The general situation in a shape matching problem is that we are given two shapes $A$ and $B$ and a certain class $T$ of allowable transformations and we want to transform $B$ optimally so that the transformed image of $B$ is as close to $A$ as possible. We assume that shapes are modeled by sets of polygonal curves. As possible class of transformations we will consider *translations*, *homotheties* (scaling and translation), *rigid motions* (rotation and translation), *similarities* (rotation, scaling and translation), and *affine maps*.

We address the problem of optimal matching of the complete shape $B$ to the complete shape $A$, called *complete-complete matching*. The algorithm we present also applies to the problem of *complete-partial matching*, i.e., matching $B$ completely as good as possible to some part of $A$, and *partial-partial matching*, i.e., matching some part of $B$ as good as possible to some part of $A$.

Several similarity measures and algorithms are known to match two curves, especially polygonal curves, see [10] for a survey. One of the “universal” similarity measures is the Hausdorff distance which is defined for any two compact sets $A$ and $B$. In [1, 3] Alt et al. describe efficient algorithms for computing the Hausdorff distance and minimizing it under translations and rigid motions for arbitrary sets of line segments. One of the drawbacks of the Hausdorff-distance is that it is very sensitive to noise. A few similarity measures are defined for pairs of curves, which capture the relative course of two curves: Fréchet distance [3], turning function distance [4], and dynamic time warping distance [6]. There are no generalizations of those distances to sets of curves, although in [2] a generalization of the Fréchet distance to geometric graphs is given, and in [9] Tanase et al. describe an algorithm for matching a set of polygonal curves to a single polygon. A similarity measure which is designed for sets of curves is the reflection visibility distance [7]. The reflection visibility distance is robust against different kinds of disturbances but is expensive to compute. Some of these similarity measures can be modified to valuate partial match, e.g., percentile-based Hausdorff distance [8].

The method we introduce is close to an intuitive notion of “matching”, i.e., find one or more candidates for the best transformations, that when applied to the shape $B$ map the most similar parts of the two shapes to each other.

## 2 Probabilistic matching

For given two shapes $A, B \subset \mathbb{R}^2$ represented by sets of polygonal curves we want to find a transformation $t$, which lets the transformed image of $B$, $t(B)$, *match best* $A$, i.e., maps the most similar parts of the shapes $A$ and $B$ to each other.

### 2.1 Simple matching algorithm

The idea of the *probabilistic approach* is quite simple:

1. Take small random samples $S_A$ from $A$ and $S_B$ from $B$ and give one “vote” to the transformation $t$ which maps $S_B$ to $S_A$.

2. Repeat this experiment many times. Then the distribution of votes in the transformation space $T$ approximates a certain probability distribution.

3. For a given neighborhood size $\delta$ take the points of $T$ with the highest number of votes in their $\delta$-neighborhood as candidates for good transformations.

The idea behind this algorithm, is that the transformations, which map large parts of shapes to each other should get significantly more votes than others. The size of the $\delta$-neighborhood influences the quality of the match.

The size of a random sample within one experiment depends on the class of transformations allowed:
For *translations*, $S_A$ and $S_B$ contain each a single randomly selected point of a correspondent shape. The transformation space is two-dimensional and the resulting translation is a vector in $\mathbb{R}^2$.

In case of *rigid motions* the transformation space is three-dimensional. A random sample of a shape within one experiment contains a random point and a

*This research was supported by the European Union under contract No. IST-511572-2, Project PROFI.

†Institute of Computer Science, Freie Universität Berlin

unit length vector corresponding to the average direction of tangent lines of its neighborhood. Two such point-vector pairs define uniquely a rigid motion.

For *similarity maps* the transformation space is four-dimensional and a random sample from a shape contains two points. Two pairs of points $a_1, a_2$ in $A$ and $b_1, b_2$ in $B$ determine a unique four-dimensional similarity transformation $t$ mapping $b_1$ to $a_1$ and $b_2$ to $a_2$.

In general, we define the $\delta$-neighborhood of a transformation $t$ as a set of transformations that map any point $b$ in $B$ into a $\delta$-neighborhood of the point $t(b)$.

## 2.2 Analysis for the Translations

As a detailed analysis shows, for most cases the translation with most votes brings the largest fitting parts of $A$ and $B$ into the $\delta$-neighborhood of each other.

The choice of $\delta$ therefore controls the trade-off between the quality of match and the size of the parts matched. With a small value of $\delta$ our algorithm would find a translation which maps nearly congruent parts of two shapes to each other, see Figure 1(a). A large value of $\delta$ leads to a translation which gives a rough match but for larger parts of the shapes, see Figure 1(b).

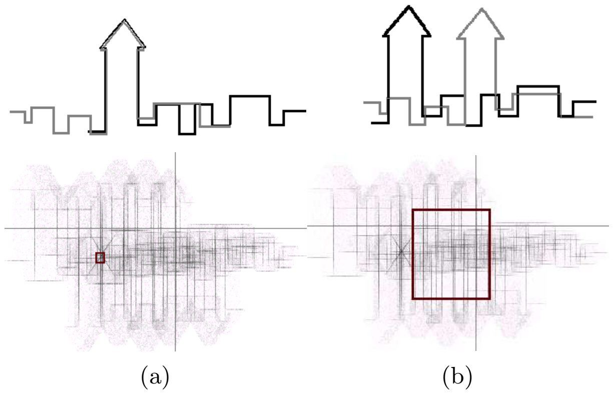

(a) (b)

Figure 1: Matching with (a) small grid size and (b) large grid size

But the value of $\delta$ does not alone determine what kind of matching we get or how large the matched parts are. For nearly congruent figures a small neighborhood size already leads to a complete-complete matching, see Figure 2(a), and with the same value for $\delta$ we can get complete-partial matching, see Figure 2(b).

Furthermore, we should examine several local maxima of the vote distribution function, since each of the maxima corresponds to a partial match between two figures, an example is illustrated in Figure 3.

For most applications it would make sense to determine for each of the candidate transformations which parts of the two shapes match and how large these

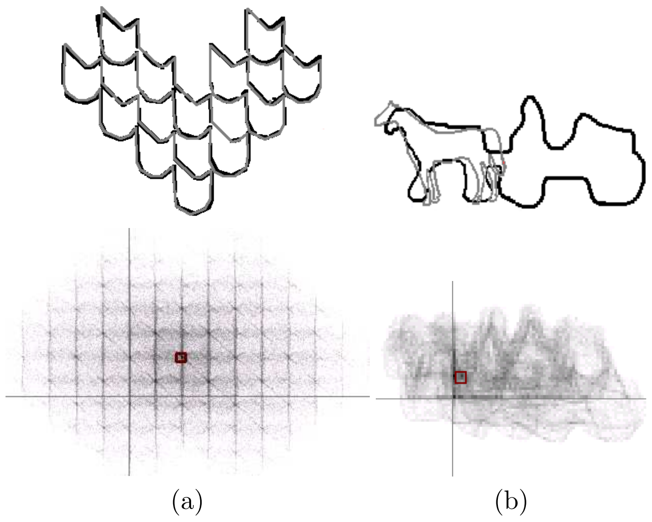

(a) (b)

Figure 2: Matching with a sampling neighborhood of the the same size in translation space for different shapes

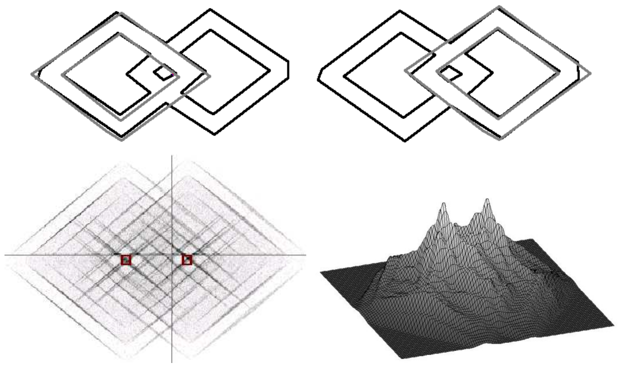


Figure 3: Top: two superimpositions of the figures each corresponding to a complete-partial match. Bottom: Experimental distribution in the translation space with two local maxima marked and a 3d-view of the distribution.

parts are. This verification step can be performed using the directed Hausdorff distance. We can incrementally take the segments of the shape $B$ into the matched set if the directed Hausdorff distance from the matched subset of $B$ to $A$ stays under the chosen value $\delta$. With the algorithm from [1] we can perform this verification step in time $O((n+m)\log(n+m))$, where $n$ and $m$ are the numbers of segments in $A$ and $B$, respectively.

The problem of partial-partial matching is not uniquely defined since there is a certain correlation between the quality of match and the size of the matched parts. We address this problem by letting the user specify the quality of match through the choice of $\delta$, for which we then find the matching parts.

**Running time.** Maximizing the the number of votes in the $\delta$-neighborhood of a translation, we maximize the measure of the set $M(t) = \{(a,b) : a \in A, b \in B, \|a - t(b)\| \leq \delta\}$. We define a similarity measure associated with a translation $t$ as $f(t) = |M(t)| / |A \times B|$. Let $\tilde{f}(t)$ denote the ratio of the number of sample translation vectors in the $\delta$-neighborhood of $t$ to the total number of samples. $\tilde{f}(t)$ is the estimate of $f(t)$. Using the technique described in [5] we can bound the absolute error for the estimate of the distribution of votes in a following way:

The number of sample translation vectors sufficient to get an approximation error of at most $\varepsilon$ with probability at least $1 - \eta$ is $N = \frac{16 \ln \frac{1}{\eta}}{\varepsilon^2} + 2$. With linear time preprocessing we can generate a random point of a shape modeled by $n$ line segments in time $O(\log n)$. That is, the time to generate $N$ random points on both shapes is $O(n + N \log n)$.

In order to find a translation with the highest number of votes, or the sampling points, in its $\delta$-neighborhood, we consider the arrangement of the $\delta$-neighborhoods of the sampling points. The basic observation is that if a sampling point $s$ is contained in the $\delta$-neighborhood of the translation $t$, then $t$ is also contained in the $\delta$-neighborhood of $s$. All translations in the same cell of the arrangement have the same sampling points in their $\delta$-neighborhoods. Therefore it is sufficient to traverse the arrangement and take the nodes with the highest number of sampling points whose $\delta$-neighborhoods contain this node. The arrangement of the $\delta$-neighborhoods of $N$ vectors has the complexity $O(N^2)$ and can be computed and traversed in time $O(N^2)$, which yields the following theorem:

**Theorem 1** *Given two shapes modeled by sets of line segments of complexity $n$ in the plane and parameter $\varepsilon, \eta$, $0 \leq \varepsilon, \eta \leq 1$. In time $O(n + \frac{\log \frac{1}{\eta} \log n}{\varepsilon^2} + \frac{\log^2 \frac{1}{\eta}}{\varepsilon^4})$ we can compute a translation $t_{\text{app}}$ such that $\left|\tilde{f}(t_{\text{app}}) - f(t_{\text{opt}})\right| \leq \varepsilon$ with probability at least $1 - \eta$, where $t_{\text{opt}}$ is a translation maximizing $f(t)$.*

### 2.3 A Faster Heuristic

We are also working on heuristics, in order to speed up the matching process, that is to find good transformation candidates with less experiments. Here we describe an algorithm which uses a set of pairs of corresponding points and during one random experiment iteratively extends that set until no further data are available or the samples are no longer consistent. For every set a transformation is computed and the best one is registered and gets a weight. Then we get a weighted sample of the transformation space, where the neighborhoods with large weight are likely to contain candidates for transformations resulting in a good match for the shapes.

The problem of computing preliminary transformations consists of two subproblems: one is to find correspondences between features, the other is to find a transformation that maps the corresponding features to each other.

**Finding Correspondences.** The initial part of every vote is the random choice of a single vertex of each of the two sets of polylines, and the direction for traversing the list of following vertices (also both possible directions may be processed). Starting from that pair a sequence of sets of pairs of vertices and vertex surrogates is generated.

Let $p_0$ be the randomly chosen vertex from the first set $S_1$ of planar polylines and let $p_1, \ldots, p_k$ be the succeeding vertices with respect to the randomly chosen direction. Analogously, let $q_0$ be the randomly chosen vertex from the second set $S_2$ of planar polylines and let $q_1, \ldots, q_l$ be the succeeding vertices. The pair $(p_0, q_0)$ is added to the – so far empty – sample set $s$.

In each iteration step distances from the last added pair of points to the next vertices on the corresponding polylines are computed. If the two computed distances are nearly equal, the next two vertices are taken as a corresponding pair which is added to the sample set $s$.

Otherwise a vertex surrogate is created for the polyline with a larger distance. A surrogate is a point lying on an edge of the polyline, but nevertheless is treated like a vertex. It is chosen to have the same distance to its predecessor as the corresponding two vertices of the other polyline have.

When the end of a polyline is reached, then starting from the initial pair the traversal is performed in the opposite direction.

**Calculating the Transformations.** For every new pair of vertices or vertex surrogates added to $s$ the transformation $t \in T$ is computed that minimizes the sum of the weighted squared distances $\varepsilon(t) = \sum_{(p_i,q_i) \in s} w(p_i, q_i) \|q_i - t(p_i)\|^2$ with $w(p_i, q_i)$ being half the length of the edges incident to $p_i$ and $q_j$.

For $T$ being the class of translations or rigid motions, $t$ and the vertex correspondences may be determined independently from each other. But for the classes of transformations that allow scalings (i.e. homotheties, similarities, and affine maps), the assignment of the vertices depends on that scaling.

In every iteration step it is checked whether the error introduced by the new added pair is still within tolerance bounds. If the error is too big, the traversal of the polylines is ceased, as illustrated in Figure 4: the bold polylines are traversed up to the end of the dashed parts. The transformation for which the error was farthest from the tolerance bound is weighted and handed over to the clustering algorithm (the transformation calculated for the bold polylines up to the beginning of the dashed line in the example).

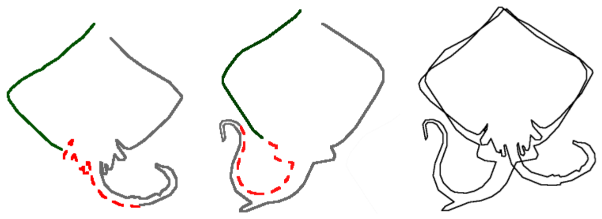


Figure 4: Two instances of the mpeg7shapeB dataset (ray-7, ray-20 and both mapped).

**Scalings:** If scalings are allowed, for every single vote a prescaling factor $c_i$ is randomly chosen such that $\mathrm{ld}(c_i)$ is normally distributed with mean value $\mathrm{ld}(\bar{c}_i)$ where $\bar{c}_i$ is the prescaling factor of the transformation rated best so far. $S_1^{c_i}$ denotes the set $S_1$ scaled by $c_i$. For the rest of the vote, every operation concerning the first set $S_1$ (e.g. computing the distance between two vertices) is performed on $S_1^{c_i}$.

**Weighting the Transformations and Clustering.** The two factors that have to be considered for weighting a transformation $t$ are the expressiveness of the sample and the quality of the match. Let $\varepsilon$ be the sum of weighted squared distances, let $w(s)$ be the sum of all weights of the pairs of points in the sample set $s$, and let $d_{bb}$ be the diameter of the bounding-box containing the covered part of the polyline. Defining the relative root mean square error $e = \sqrt{\varepsilon / w(s)}/d_{bb}$ yields a value representing the quality of the match which is invariant under scalings.

The match score or weight $w(t)$ of a transformation $t$ is then defined as $w(t) = l/(1 + \gamma \cdot e)$ with $\gamma$ being an arbitrarily chosen constant for balancing out the impact of the length $l$ and the error $e$.

Let $t_1$ and $t_2$ be two arbitrary transformations and let $S$ be the transformed shape. The distance measure $d_S(t_1, t_2) = \max_{p \in S'} \|t_1(p) - t_2(p)\|$, with $S'$ being the set of the vertices of the bounding box of $S$ forms a metric space for affine maps, under the assumption that the four points of $S'$ are pairwise different.

A cluster in our sense represents a region of limited diameter, which subsumes a considerable amount of weights of the enclosed input points (transformations).

Let $T_n$ be the set of $n$ preliminary transformations and $w_i$ be the weight of transformation $t_i \in T_n$. For every transformation $t_k$ the set of its dominators in range $r$ is defined as a set of transformations with distance at most $r$ to $t_k$ with a weight greater than $w_k$, which have no dominators in range $r$. Every transformation $t_c$ which has no dominators in range $r_c$ defines a cluster with radius $r_c$ as the set $\{t_i \in T_n | d(t_c, t_i) < r_c\}$, i.e., a transformation is either assigned to its dominators' clusters or it defines a cluster itself. The weight of a cluster is defined as the sum of the weights of its elements. This definition allows for a fast computation of all clusters and their weights.

## 3 Conclusion

We presented a probabilistic approach for matching two shapes represented by sets of polygonal curves, which is close to the human notion of match and is easy to implement. The algorithms are also applicable to the problem of partial matching and we obtained convincing results from experiments with the MPEG7 shape dataset.

## References

[1] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13:251–265, 1995.

[2] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. of Algorithms*, pages 262–283, 2003.

[3] H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of computational geometry*, pages 121 – 153. Elsevier Science Publishers B.V. North-Holland, 1999.

[4] E. Arkin, P. Chew, D. Huttenlocher, K. Kedem, and J. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–215, March 1991.

[5] O. Cheong, A. Efrat, and S. Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1091–1100, 2004.

[6] A. Efrat and S. Venkatasubramanian. Curve matching, time warping, and light fields. Technical Report AT&T TD-4z5TMU, AT&T.

[7] M. Hagedoorn, M. Overmars, and R. Veltkamp. A new visibility partition for affine pattern matching. In *Proc. Discrete Geometry for Computer Imagery conference, DGCI 2000*, pages 358–370, Berlin, 2000. Springer-Verlag.

[8] M. Hagedoorn and R. Veltkamp. State-of-the-art in shape matching. Technical Report UU-CS-1999-27, Utrecht University, the Netherlands, 1999.

[9] M. Tanase, R. C. Veltkamp, and H. Haverkort. Multiple polyline to polygon matching. In *Proceedings of the 16th Annual Symposium on Algorithms and Computation (ISAAC 2005), LNCS 3827*, pages 60–70, 2005.

[10] R. C. Veltkamp. Shape matching: Similarity measures and algorithms. Technical Report UU-CS-2001-03, Utrecht University, 2001.

# On the ICP Algorithm[*]

Esther Ezra[†] Micha Sharir[‡] Alon Efrat[§]

## Abstract

We present upper and lower bounds for the number of iterations performed by the Iterative Closest Point (ICP) algorithm. This algorithm has been proposed by Besl and McKay [4] as a successful heuristics for pattern matching under translation, where the input consists of two point sets in $d$-space, for $d \geq 1$, but so far it seems not to have been rigorously analyzed. The considered (standard) measure of resemblance that the algorithm attempts to optimize is the RMS (root mean squared distance). We show that the number of iterations performed by the algorithm is polynomial in the number of input points. In particular, this bound is quadratic in the one-dimensional problem, for which we present a lower bound construction of $\Omega(n \log n)$ iterations, where $n$ is the overall size of the input. We also present several structural geometric properties of the algorithm. We show that at each iteration of the algorithm the cost function monotonically and strictly decreases along the vector $\Delta t$ of the *relative translation.* As a result, we conclude that the polygonal path $\pi$, obtained by concatenating all the relative translations that are computed during the execution of the algorithm, does not intersect itself. In particular, in the one-dimensional problem all the relative translations of the ICP algorithm are in the same (left or right) direction.

## 1 Introduction

The matching and analysis of geometric patterns and shapes is an important problem that arises in various application areas, in particular in computer vision and pattern recognition [3]. In a typical scenario, we are given two objects $A$ and $B$, and we wish to determine how much they *resemble* each other. Usually one of the objects may undergo certain transformations, like translation, rotation and/or scaling, in order to be matched with the other object as well as possible. In many cases, the objects are represented as finite sets of (sampled) points in two or three dimensions (they are then referred to as "point patterns" or "shapes"). In order to measure "resemblance", various *cost functions* have been used. A prominent one is the *sum of squared distances* or *root mean square* [4, 5], Under this measure, the cost function is $\Phi_2(A,B) = \frac{1}{m}\sum_{a\in A}\|a - N_B(a)\|^2$, where $N_B(a)$ denotes the nearest neighbor of $a$ in $B$, and where $m = |A|$. In what follows, we also use the (slightly abused) notation

$$RMS(t) := \frac{1}{m}\sum_{a\in A}\|a + t - N_B(a+t)\|^2. \qquad (1)$$

A heuristic matching algorithm that is widely used, due to its simplicity (and its good performance in practice), is the *Iterative Closest Point* algorithm, or the *ICP* algorithm for short, of Besl and McKay [4]. Given two point sets $A$ and $B$ in $\mathbb{R}^d$ (also referred to as the *data shape* and the *model shape*, respectively), we wish to minimize a cost function $\phi(A+t, B)$, over all *translations* $t$ of $A$ relative to $B$. The algorithm starts with an arbitrary translation that aligns $A$ to $B$ (suboptimally), and then repeatedly performs local improvements that keep re-aligning $A$ to $B$, while decreasing the given cost function $\phi(A+t,B)$, until a convergence is reached. This is done as follows.

At the $i$-th iteration of the ICP algorithm, the set $A$ has already been translated by some vector $t_{i-1}$, where $t_0 = \overrightarrow{0}$. We then apply the following two steps:

(i) We assign each (translated) point $a + t_{i-1} \in A + t_{i-1}$ to its nearest neighbor $b = N_B(a+t_{i-1}) \in B$ under the Euclidean distance[1]. (ii) We then compute the new relative translation $\Delta t_i$ that minimizes the cost function $\phi$ (with respect to the above fixed assignment). Specifically, we find the $\Delta t_i$ that minimizes

$$\phi_2(A + t_{i-1}, \Delta_{t_i}, B) = \frac{1}{m}\sum_{a\in A}\|a + t_{i-1} + \Delta t_i - N_B(a + t_{i-1})\|^2.$$

We then align the points of $A$ to $B$ by translating them by $\Delta t_i$, so the new (overall) translation is $t_i = t_{i-1} + \Delta t_i$.

The ICP algorithm performs these two steps repeatedly and stops when the value of the cost function does

[*]Work on this paper by the first two authors has been supported by NSF Grants CCR-00-98246 and CCF-05-14079, by a grant from the U.S.-Israeli Binational Science Foundation, by Grant 155/05 from the Israel Science Fund, Israeli Academy of Sciences, by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. Work on this paper by the last author has been partially supported by an NSF CAREER award (CCR-0348000) and an ITR/Collaborative Research grant (0312443).

[†]Department of Computer Science, University of Tel-Aviv, estere@post.tau.ac.il

[‡]Department of Computer Science, University of Tel-Aviv, michas@post.tau.ac.il

[§]Department of Computer Science, University of Arizona, alon@cs.arizona.edu

[1]Of course, other distances can also be considered, but this paper treats only the Euclidean case.

not decrease with respect to the previous step (as a matter of fact, the ICP algorithm in its original presentation stops when the difference in the cost function falls below a given threshold $\tau > 0$; however, in our analysis, we assume that $\tau = 0$). It is shown by Besl and McKay [4] that, when $\phi(\cdot,\cdot)$ measures the sum of squared distances, this algorithm always converges monotonically to a local minimum, and that the value of the cost function decreases at each iteration[2].

In other words, in stage (i) of each iteration of the ICP algorithm we assign the points in (the current translated copy of) $A$ to their respective nearest neighbors in $B$, and in stage (ii) we translate the points of $A$ in order to minimize the value of the cost function with respect to the assignment computed in stage (i). This in turn may cause some of the points in the new translated copy of $A$ to acquire new nearest neighbors in $B$, which causes the algorithm to perform further iterations. If no point of $A$ changes its nearest neighbor in $B$, the value of the cost function does not change in the next iteration (in fact, the next relative translation equals $\overrightarrow{0}$) and, as a consequence, the algorithm terminates. Note that the pattern matching performed by the algorithm is *one-directional*, that is, it aims to find a translation of $A$ that places the points of $A$ near points of $B$, but not necessarily the other way around.

Since the value of the cost function is strictly reduced at each iteration of the algorithm, it follows that no nearest-neighbor assignment arises more than once during the course of the algorithm, and thus it is sufficient to bound the overall number of nearest-neighbor assignments (or, NNA's, for short) that the algorithm reaches in order to bound the number of its iterations.

## 2 General Structural Properties of the ICP Algorithm under the RMS Measure

Let $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$ be two point sets in $d$-space, for $d \geq 1$, and suppose that the ICP algorithm aligns $A$ to $B$; that is, $B$ is fixed and $A$ is translated to best fit $B$.

**Theorem 1** *The maximum possible overall number of nearest-neighbor assignments, over all translated copies of $A$, is $\Theta\left(m^d n^d\right)$.*

**Sketch of proof**: Let $\mathcal{V}(B)$ denote the *Voronoi diagram* of $B$, that is, the partition of $\mathbb{R}^d$ into $d$-dimensional cells $\mathcal{V}(b_i)$, for $i = 1, \ldots, n$, such that each point $p \in \mathcal{V}(b_i)$ satisfies $\|p - b_i\| \leq \|p - b_j\|$, for each $j \neq i$.

The global NNA changes at critical values of the translation $t$, in which the nearest-neighbor assignment of some point $a + t$ of the translated copy of $A$ is changed; that is, $a$ crosses into a new Voronoi cell of $\mathcal{V}(B)$. For each $a \in A$ (this denotes the *initial* location of this point) consider the shifted copy $\mathcal{V}(B) - a = \mathcal{V}(B - a)$ of $\mathcal{V}(B)$; i.e., the Voronoi diagram of $B - a = \{b - a \mid b \in B\}$. Then a critical event that involves the point $a_i$ occurs when the translation $t$ lies on the boundary of some Voronoi cell of $\mathcal{V}(B - a_i)$, for $i = 1, \ldots, m$. Hence we need to consider the *overlay* $M(A, B)$ of the $m$ shifted diagrams $\mathcal{V}(B - a_1), \ldots, \mathcal{V}(B - a_m)$. Each cell of the overlay consists of translations with a common NNA, and the number of assignments is in fact equal to the number of cells in the overlay $M(A, B)$. A recent result of Koltun and Sharir [6] implies that the complexity of the overlay is $O(m^d n^d)$. It is straightforward to give constructions that show that this bound is tight in the worst case, for any $d \geq 1$. □

[2]We definitely decrease it with respect to the present nearest-neighbor assignment, and the revised nearest-neighbor assignment at the new placement can only decrease it further.

**Corollary 2** *For any cost function that guarantees convergence (in the sense that the algorithm does not reach the same NNA more than once), the ICP algorithm terminates after $O(m^d n^d)$ iterations.*

**Remark:** A major open problem is to determine whether this bound is tight in the worst case. So far we have been unable to settle this question (under the RMS measure) even for $d = 1$; see below for details. In other words, while there can be many NNA's, we suspect that the ICP algorithm cannot step through many of them in a single execution.

We next present a simple but crucial property of the relative translations that the algorithm generates.

**Lemma 3** *At each iteration $i \geq 2$ of the algorithm, the relative translation vector $\Delta t_i$ satisfies*

$$\Delta t_i = \frac{1}{m}\left(\sum_{a \in A} N_B(a + t_{i-1}) - N_B(a + t_{i-2})\right), \qquad (2)$$

*where $t_j = \sum_{k=1}^{j} \Delta t_k$.*

**Proof**: Follows using easy algebraic manipulations, based on the obvious equality that follows by construction

$$\Delta t_i = \frac{1}{m}\left(\sum_{a \in A}(N_B(a + t_{i-1}) - (a + t_{i-1}))\right).$$

□

**Remark:** The expression in (2) only involves differences between points of $B$. More precisely, the next relative translation is the average of the differences between the new $B$-nearest neighbor and the old $B$-nearest neighbor of each point of (the current and preceding translations of) $A$. This property does not hold for the *first* relative translation of the algorithm.

**Theorem 4** *Let $\Delta t$ be a move of the ICP algorithm from translation $t_0$ to $t_0 + \Delta t$. Then $RMS(t_0 + \xi\Delta t)$ is a strictly decreasing function of $\xi \in [0, 1]$.*

**Proof:** The function (see also (1) )

$$RMS(t) = \frac{1}{m}\sum_{a \in A}\left(\|t\|^2 + 2t\cdot(a - N_B(a+t)) + \|a - N_B(a+t)\|^2\right)$$

is the average of $m$ Voronoi surfaces $\mathcal{S}_{B-a}(t)$, whose respective minimization diagrams are $\mathcal{V}(B - a)$, for each $a \in A$. That is,

$$\mathcal{S}_{B-a}(t) = \min_{b \in B}\|a + t - b\|^2 = \min_{b \in B}\left(\|t\|^2 + 2t\cdot(a - b) + \|a - b\|^2\right),$$

for each $a \in A$. Subtracting the term $\|t\|^2$, we obtain that each resulting Voronoi surface $\mathcal{S}_{B-a}(t) - \|t\|^2$ is the

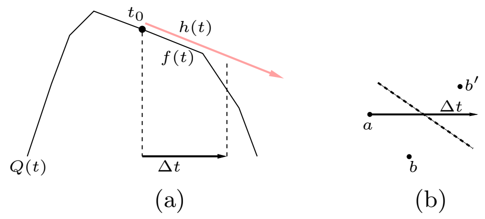


Figure 1: (a) Illustrating the proof that $RMS(t_0 + \xi\Delta t)$ is a strictly decreasing function of $\xi \in [0, 1]$. (b) The new nearest neighbor lies ahead of the old one in the direction $\Delta t$

lower envelope of $n$ hyperplanes, and is thus the boundary of a concave polyhedron. Hence $Q(t) := RMS(t) - \|t\|^2$ is equal to the average of these concave polyhedral functions, and is thus the boundary of a concave polyhedron (see also the proof of Theorem 1).

Consider the NNA that corresponds to the translation $t_0$. It defines a facet $f(t)$ of $Q(t)$, which contains the point $(t_0, Q(t_0))$. We now replace $f(t)$ by the hyperplane $h(t)$ containing it, and note that $h(t)$ is tangent to the polyhedron $Q(t)$ at $t_0$; see Figure 1(a) for an illustration. Put $RMS_0(\xi) := \frac{1}{m}\sum_{a\in A} \|a + t_0 + \xi\Delta t - N_B(a + t_0)\|^2$. The graph of $RMS_0(\xi)$ is the image of the relative translation vector $\Delta t$ on the paraboloid $\|t\|^2 + h(t)$. Since $Q(t) \le h(t)$, for any $t \in \mathbb{R}^d$, the concavity of $Q(t)$ implies that for any $0 \le \xi_1 < \xi_2 \le 1$, $Q(t_0 + \xi_1\Delta t) - Q(t_0 + \xi_2\Delta t) \ge h(t_0 + \xi_1\Delta t) - h(t_0 + \xi_2\Delta t)$. Since $\|t\|^2 + h(t)$ is (strictly) monotone decreasing[3] along $\Delta t$, we obtain

$$RMS(t_0 + \xi_1\Delta t) - RMS(t_0 + \xi_2\Delta t) =$$

$$\|t_0+\xi_1\Delta t\|^2 + Q(t_0+\xi_1\Delta t) - \|t_0+\xi_2\Delta t\|^2 - Q(t_0+\xi_2\Delta t) \ge$$

$$\|t_0+\xi_1\Delta t\|^2 - \|t_0+\xi_2\Delta t\|^2 + h(t_0+\xi_1\Delta t) - h(t_0+\xi_2\Delta t) > 0,$$

which implies that $RMS(t_0 + \xi\Delta t)$ is a strictly decreasing function of $\xi \in [0, 1]$. □

Let $\pi$ be the connected polygonal path obtained by concatenating the ICP relative translations $\Delta t_j$. That is, $\pi$ starts at the origin and its $j$-th edge is the vector $\Delta t_j$. Theorem 4 implies:

**Theorem 5** *The ICP path $\pi$ does not intersect itself.*

**Corollary 6 (Monotonicity)** *In the one-dimensional case, the ICP algorithm moves the points of $A$ always in the same (left or right) direction. That is, either $\Delta t_i \ge 0$ for each $i \ge 0$, or $\Delta t_i \le 0$ for each $i \ge 0$.*

**Corollary 7** *In any dimension $d \ge 1$, the angle between any two consecutive edges of $\pi$ is obtuse.*

**Proof:** Consider two consecutive edges $\Delta t_k$, $\Delta t_{k+1}$ of $\pi$. Using Lemma 3 we have

$$\Delta t_{k+1} = \frac{1}{m}\sum_{a\in A}\Big(N_B(a + t_k) - N_B(a + t_{k-1})\Big).$$

[3]By definition, $\Delta t$ moves from $t_0$ to the minimum of the fixed paraboloid $\|t\|^2 + h(t)$, whence the claim.

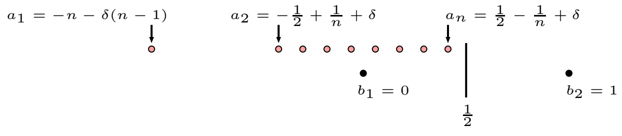

$a_1 = -n - \delta(n-1)$
$a_2 = -\frac{1}{2} + \frac{1}{n} + \delta$
$a_n = \frac{1}{2} - \frac{1}{n} + \delta$
$b_1 = 0$
$\frac{1}{2}$
$b_2 = 1$

Figure 2: The lower bound construction. Only the two leftmost cells of $\mathcal{V}(B)$ are depicted.

It is easy to see that (consult Figure 1(b))

$$\Big(N_B(a + t_k) - N_B(a + t_{k-1})\Big) \cdot \Delta t_k \ge 0,$$

for each $k \ge 1$, where equally holds if and only if $a$ does not change its $B$-nearest neighbor. Hence $\Delta t_{k+1} \cdot \Delta t_k \ge 0$. It is easily checked that equality is possible only after the last step (where $\Delta t_{k+1} = 0$). □

## 3 The ICP Algorithm on the Line under the RMS Measure

In this section we consider the special case $d = 1$, and analyze the performance of the ICP algorithm on the line under the RMS measure. Theorem 1 implies that in this case the number of NNA's, and thus the number of iterations of the algorithm, is $O(mn)$. In general, we do not know whether this bound is sharp in the worst case (we strongly believe that it is not). However, in the worst case, the number of iterations can be superlinear:

**Theorem 8** *There exist point sets $A$, $B$ on the real line of arbitrarily large common size $n$, for which the number of iterations of the ICP algorithm (under the RMS measure) is $\Theta(n \log n)$.*

**Proof:** We construct two point sets $A$, $B$ on the real line, where $|A| = |B| = n$. The set $A$ consists of the points $a_1 < \cdots < a_n$, where $a_1 = -n - \delta(n-1)$, $a_i = \frac{2(i-1)-n}{2n} + \delta$, for $i = 2, \ldots, n$, and $\delta = o\left(\frac{1}{n}\right)$ is some sufficiently small parameter. The set $B$ consists of the points $b_i = i - 1$, for $i = 1, \ldots, n$. See Figure 2.

Initially, all the points of $A$ are assigned to $b_1$. As the algorithm progresses, it keeps translating $A$ to the right. The first translation satisfies

$$\Delta t_1 = \frac{1}{n}\sum_{i=1}^{n}(b_1 - a_i) = \frac{1}{n}(b_1 - a_1) - \frac{n-1}{n}\delta = 1,$$

which implies that after the first iteration of the algorithm all the points of $A$, except for its leftmost point, are assigned to $b_2$. Using (2), we have $\Delta t_2 = \frac{1}{n}\sum_{i=1}^{n-1}(b_2 - b_1) = \frac{n-1}{n}$, which implies that the $n-1$ rightmost points of $A$ move to the next Voronoi cell $\mathcal{V}(b_3)$ after the second iteration, so that the distance between the new position of $a_n$ from the right boundary of $\mathcal{V}(b_3)$ is $\frac{2}{n} - \delta$, and the distance between the new position of $a_2$ and the left boundary of $\mathcal{V}(b_3)$ is $\delta$, as is easily verified.

In the next iteration $\Delta t_3 = \frac{n-1}{n}$ (arguing as above). However, due to the current position of the points of $A$ in $\mathcal{V}(b_3)$, only the $n-2$ rightmost points of $A$ cross the right Voronoi boundary of $\mathcal{V}(b_3)$ (into $\mathcal{V}(b_4)$), the nearest neighbor of $a_2$ remains unchanged (equal to $b_3$).

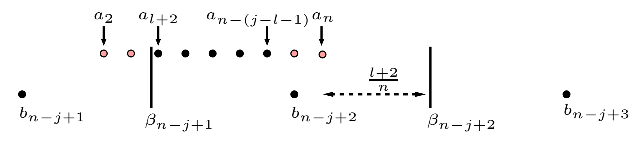


Figure 3: At the last iteration of round $j$, after shifting the points of $A$ by $\Delta t = \frac{j}{n}$ to the right, the points $a_{l+2}, \ldots, a_{n-(j-l-1)}$ (represented in the figure as black bullets) still remain in $\mathcal{V}(b_{n-j+2})$.

We next show, using induction on the number of Voronoi cells the points of $A$ have crossed so far, the following property. Assume that the points of $A$, except for the leftmost one, are assigned to $b_{n-j+1}$ and $b_{n-j+2}$, for some $1 \le j \le n$ (clearly, these assignments can involve only two consecutive Voronoi cells), and consider all iterations of the algorithm, in which some points of $A$ cross the common Voronoi boundary $\beta_{n-j+1}$ of the cells $\mathcal{V}(b_{n-j+1})$, $\mathcal{V}(b_{n-j+2})$. Then, (i) at each such iteration the relative translation is $\frac{j}{n}$, (ii) at each iteration, other than the last one, the overall number of points of $A$ that cross $\beta_{n-j+1}$ is exactly $j$, and no point crosses any other boundary, and (iii) at the last iteration of the round, the overall number of points of $A$ that cross either $\beta_{n-j+1}$ or $\beta_{n-j+2}$ is exactly $j-1$. In fact, in the induction step we assume that properties (i), (ii) hold, and, as a consequence, show that property (iii) follows.

To prove this property, we first note, using (2), that the relative translation at each iteration of the algorithm is $\frac{k}{n}$, for some integer $1 \le k \le n$. The preceding discussion shows that the induction hypothesis holds for $j = n$ and $j = n-1$. Suppose that it holds for all $j' \ge j$, for some $2 \le j \le n-1$, and consider round $(j-1)$ of the algorithm, during which points of $A$ cross $\beta_{n-j+2}$ (that is, we consider all iterations with that property). Thus, at each iteration of round $j$ (except for the last one), in which there are points of $A$ that remain in the cell $\mathcal{V}(b_{n-j+1})$, the $j$ rightmost points of $A$ (among those contained in $\mathcal{V}(b_{n-j+1})$) cross $\beta_{n-j+1}$. Let us now consider the last such iteration. In this case, all the points of $A$, except $l$ of them, for some $0 \le l < j$ (and the leftmost point, which we ignore), have crossed $\beta_{n-j+1}$ (in previous iterations). The key observation is that the distance from the current position of $a_n$ to the next Voronoi boundary $\beta_{n-j+2}$ is $\frac{l+2}{n} - \delta$ (this follows since we shift in total $n-1$ points of $A$ that are equally spaced by $\frac{1}{n}$), and since the next translation $\Delta t$ satisfies $\Delta t = \frac{j}{n}$ (using the induction hypothesis and (2)), it follows that only $j-1$ points of $A$ cross a Voronoi boundary in the next iteration. Moreover, the points $a_2, \ldots, a_{l+1}$ cross the boundary $\beta_{n-j+1}$, and the points $a_{n-(j-l-2)}, \ldots, a_n$ cross the boundary $\beta_{n-j+2}$ (this is the first move in which this boundary is crossed at all); see Figure 3 for an illustration.

Thus, at the next iteration, since only $j-1$ points have just crossed between Voronoi cells, (2) implies that the next translation is $\frac{j-1}{n}$, and, as is easily verified, at each further iteration, as long as there are at least $j-1$ points of $A$ to the left of $\beta_{n-j+2}$, this property must continue to hold, and thus $j-1$ points will cross $\beta_{n-j+2}$. This establishes the induction step.[4]

It now follows, using the above properties, that the number of iterations required for all the points of $A$ to cross $\beta_{n-j+1}$ is $\lceil \frac{n}{j} \rceil$, where in the first (last) such iteration some of the points may cross $\beta_{n-j}$ ($\beta_{n-j+2}$) as well. This implies that the number of such iterations, in which the points of $A$ cross only $\beta_{n-j+1}$ (and none of the two neighboring Voronoi boundaries), is at least $\left\lceil \frac{n}{j} \right\rceil - 2$ (but not more than $\lceil n/j \rceil$). Thus the overall number of iterations of the algorithm is $\Theta\left(\sum_{j=1}^{n} \left\lceil \frac{n}{j} \right\rceil\right) = \Theta(n \log n)$. $\square$

## 4 Concluding Remarks

A major open problem that this paper raises is to improve the upper bound, or, alternatively, present a tight lower bound construction on the number of iterations performed by the algorithm. This problem is challenging even in the one-dimensional case. So far, we have not managed to obtain a construction that yields $\Omega(n^2)$ iterations, and we conjecture that the actual bound is subquadratic in this case, perhaps matching our lower bound, i.e., $\Theta(n \log n)$.

Finally, we note that some of the results given in this paper were supported and verified by running experimentation. Our implementation is based on the CGAL [1] and LEDA [2] libraries.

**Acknowledgments.** The authors wish to thank Boris Aronov for useful discussions concerning the problem. In particular, during these discussions, the proof of Theorem 4 has been obtained, and the parametrization given in the construction presented in Section 3 has been simplified (from the original construction given by the authors). The authors also wish to thank Leo Guibas, Pankaj Agarwal, Jie Gao and Vladlen Koltun for helpful discussions concerning this problem. In particular, a preliminary version of the proof of Theorem 4 has been obtained during these discussions.

## References

[1] The CGAL project homepage. `http://www.cgal.org/`.

[2] The LEDA homepage. `http://www.algorithmic-solutions.com/enleda.htm`.

[3] H. Alt and L. Guibas. Discrete geometric shapes: matching, interpolation, and approximation. In Handbook of Computational Geometry. J.-R. Sack and J. Urrutia eds. Elsevier, Amsterdam, pages 121-153, 1999.

[4] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[5] S. Har-Peled and B. Sadri. How fast is the $k$-means method? *Algorithmica*, 41(3):185–202, 2005.

[6] V. Koltun, and M. Sharir. On overlays of minimization diagrams. Manuscript, 2005.

[4]Note that the induction step first establishes property (iii) of the preceding round, and then (i) and (ii) of the current round.

# Noisy disk set matching under rigid motion

Yago Diez and J. Antoni Sellarès*

**Abstract**

Let $\mathcal{A}$ and $\mathcal{B}$ be two disk sets, with $|\mathcal{A}| \leq |\mathcal{B}|$. We propose a process for determining matches between $\mathcal{A}$ and subsets of $\mathcal{B}$ under rigid motion, assuming that the position of all disks in both sets contains a certain amount of "noise". The process consists on two main stages: a candidate zone determination algorithm and a matching algorithm. A candidate zone is a region determined by one, two or four squares that contains a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\mathcal{A}$ may match one or more subsets $\mathcal{B}'$ of $\mathcal{S}$. We use a compressed quadtree to have easy access to the subsets of $\mathcal{B}$ related to candidate zones. In each quadtree node we store geometric information that is used by the algorithm that searches for candidate zones. The second algorithm solves the disk set matching problem: we generate all, up to a certain equivalence, possible motions that bring $\mathcal{A}$ close to some subset $\mathcal{B}'$ of every $\mathcal{S}$ and seek for a matching between sets $\mathcal{A}$ and $\mathcal{B}'$.

## 1 Introduction

Determination of the presence of a geometric pattern in a large set of objects is a fundamental problem in computational geometry. It arises in diverse applications such as astronomy (constellation recognition problem) and molecular biology (substructure search problem). Stars can be seen as disks in $\mathbb{R}^2$ with radii determined by their brightness and an atom in a protein molecule can be modelled as a ball in $\mathbb{R}^3$ whose radius is the Van Der Waals radius of the element it represents. Since star and atom positions are fuzzy, both problems can be transformed to approximate disk/ball set matching under rigid motion problems. In this paper we will concentrate in the two-dimensional case.

### 1.1 Problem formulation

A *rigid motion* in $\mathbb{R}^2$ is a distance preserving mapping that can be expressed as a composition of a rotation and a translation. Fixed a real number $\epsilon \geq 0$ we say that two disks $D(a,r)$, $D(b,s)$ of centers $a, b$ and radii $r, s$ *approximately match* when $r = s$ and $d(a,b) \leq \epsilon$, where $d$ denotes the Euclidean distance.

Let $\mathcal{D}$, $\mathcal{S}$ be two disk sets of the same cardinality. A *radius preserving bijective mapping* $f : \mathcal{D} \to \mathcal{S}$ maps each disk $A = D(a,r) \in \mathcal{D}$ to a distinct and unique disk $f(A) = D(b,s) \in \mathcal{S}$ so that $f(a) = b$ and $r = s$. Let $\mathcal{F}$ be the set of all radius preserving bijective mappings between $\mathcal{D}$ and $\mathcal{S}$. Observe that when the disks in $\mathcal{D}$ and $\mathcal{S}$ have a high number of different radii, $|\mathcal{F}|$ may severely diminish. The *bottleneck distance* between $\mathcal{D}$ and $\mathcal{S}$ is defined as:

$$d_b(\mathcal{D},\mathcal{S}) = \min_{f\in\mathcal{F}} \max_{A\in\mathcal{D}} d(A, f(A)) .$$

The **Noisy Disk Matching (NDM)** problem can be formulated as follows. Given two disk sets $\mathcal{A}$, $\mathcal{B}$, $|\mathcal{A}| \leq |\mathcal{B}|$, and $\epsilon \geq 0$, determine all rigid motions $\tau$ for which there exists a subset $\mathcal{B}'$ of $\mathcal{B}$ such that $d_b(\tau(\mathcal{A}), \mathcal{B}') \leq \epsilon$.

If $\tau$ is a solution to the **NDM** problem, every disk of $\tau(\mathcal{A})$ approximately matches to a distinct and unique disk of $\mathcal{B}'$ of the same radius, and we say that $\mathcal{A}$ and the subset $\mathcal{B}'$ of $\mathcal{S}$ are *noisy congruent*. According to our initial motivation, we are only interested in subsets $\mathcal{B}'$ so that $\mathcal{A}$ and $\mathcal{B}'$ are noisy congruent, and not so much in the individual matchings between disks in $\tau(\mathcal{A})$ and $\mathcal{B}'$.

If we think of a point as a disk of 0 radius and point sets of the same cardinality are considered, then the **NDM** problem becomes the **Noisy Matching (NM)** problem: Given two point sets $\mathcal{A}$, $\mathcal{B}$ of the same cardinality and $\epsilon \geq 0$, determine, if possible, a rigid motion $\tau$ such that $d_b(\tau(\mathcal{A}), \mathcal{B}) \leq \epsilon$ .

### 1.2 Previous Results

The study of the **NM** problem was initiated by Alt *et al.* [1] who presented an exact $O(n^8)$ time algorithm for solving the problem for two sets of cardinality $n$. This bound can be reduced to $O(n^3 \log n)$ if an assignment of points in $\mathcal{A}$ to points in $\mathcal{B}$ is given [1]. Combining Alt et alt. algorithm with the techniques by Efrat *et al.* [3] the time can be reduced to $O(n^7 \log n)$.

We must note here that none of these approaches considers the possibility of working with disk sets and that the fact of having sets of different cardinality is often not considered.

## 2 Our Approach

Our main goal is to discretize the **NMD** problem by turning it into a series of "smaller" instances, expecting that their solution will be faster. To do so, we will use a conservative strategy to discard those subsets of $\mathcal{B}$ where no noisy match may happen and keep a number of zones where this matches may occur.

*Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain, {ydiez,sellares}@ima.udg.es. Partially supported by grant TIN2004-08065-C02-02.

We will assume that all rectangles and squares we consider are axis-parallel. Our algorithm consists on two main parts. The first one yields a collection of *candidate zones*, which are regions determined by one, two or four squares that contain a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\mathcal{A}$ may approximately match one or more subsets $\mathcal{B}'$ of $\mathcal{S}$. The second part of the algorithm solves the **NDM** problem between $\mathcal{A}$ and every $\mathcal{B}'$.

The discarding decisions throughout the first part of this process will be made according to a series of geometric parameters, invariant under rigid motion, that will help us to describe the shapes of $\mathcal{A}$ and the different subsets of $\mathcal{B}$ that we explore. To navigate $\mathcal{B}$ and have easy access to those subsets, we will use a *compressed quadtree* [2]. By doing this we intend to achieve a reduction of the total computational time, corresponding to a pruning of the search space, as an effect of all the calculations we avoid by discarding parts of $\mathcal{B}$ cheaply and at an early stage.

The candidate zone determination algorithm consists itself of two subparts: a quadtree construction algorithm and a search algorithm that traverses the quadtree looking for the candidate zones. The quadtree construction algorithm can also be subdivided in two more parts: a compressed quadtree building algorithm that uses the centers of the disks in $\mathcal{B}$ as sites (without considering their radii), and then an algorithm that adds the information related to the geometric parameters being used to each node.

The second part of the algorithm consists on two more parts. The first one, the "enumeration" part, will group all possible rigid motions of $\mathcal{A}$ in equivalence classes in order to make their handling feasible. We will choose a representative motion $\tau$ for every equivalence class. The second step, the "testing" part, will perform a bipartite matching algorithm between every set $\tau(\mathcal{A})$ and every disk set $\mathcal{B}'$ associated to a candidate zone. For these matching tests we will modify the algorithm proposed in [3] by using the *skip-quadtree* data structure [4] in order to make it easier to implement and to take advantage of the data structures that we have already built.

## 3 Candidate zone determination algorithm

Let $R_{\mathcal{A}}$ be the minimal rectangle that contains all the centers of the disks in $\mathcal{A}$, and let $s$ be the smallest positive integer for which $(\text{diagonal}(R_{\mathcal{A}})+2\epsilon) \leq 2^s$ holds. It is not difficult to prove that for any rigid motion $\tau$ there exists an square of *size* $s$ (with side length $2^s$) containing all the centers in $\tau(\mathcal{A})$. This allows us to affirm that, for any $\mathcal{S} \subset \mathcal{B}$ noisy congruent with $\mathcal{A}$ there will exist a square of size $s$ that contains the centers of its disks. In this first step of our algorithm, instead of looking for all possible rigid motions of set $\mathcal{A}$ we will look for such squares. More specifically, we will store the centers of the disks in $\mathcal{B}$ in a compressed PR quadtree $\mathcal{Q}_{\mathcal{B}}$ and describe the geometry of each of the nodes in this quadtree using a number of geometric parameters that are invariant for rigid motions. Then we will look for candidate zones in the quadtree whose associated geometric parameters match those of $\mathcal{A}$. Although our intention would be to describe our candidate zones exactly as squares of size $s$ this will not always be possible, so we will also have to use two or four squares of size $s$. It is important to stress the fact that ours is a conservative algorithm, so we will not so much look for candidate zones as rule out those regions where no candidate zones may appear.

### 3.1 Compressed quadtree construction

Although for the first part of the algorithm we will only use the quadtree levels between the root and the one whose associated nodes have size $s$, we will use the remaining levels later, so we build the whole compressed quadtree $\mathcal{Q}_{\mathcal{B}}$. We use the techniques in [2] to ensure a total asymptotic cost of $O(m \log m)$ in all cases, where $m = |\mathcal{B}|$.

#### 3.1.1 Adding information to the quadtree

To simplify explanations we will consider $\mathcal{Q}_{\mathcal{B}}$ to be complete. Although it is clear that this will not be the general situation this limitation can be easily overcome in all the parts of the algorithm.

At this moment the quadtree $\mathcal{Q}_{\mathcal{B}}$ contains no information about the different radii of the disks in $\mathcal{B}$ or the geometric characteristics of $\mathcal{B}$ as a whole. Since these parameters will guide our search for matches they must be invariant under rigid motion. Some examples of geometric parameters we can consider are: a) parameters that take into account the fact that we are working with disk sets: number of disks or list of disk's radii attached to a node; b) parameters based on distances between centers: maximum and minimum distance between centers or maintaining the whole of the distance matrix depending on our practical memory requirements and the performance we achieve. For every geometric parameter we will define a *parameter compatibility criterium* that will allow us to discard zones of the plane that cannot contain a subset $\mathcal{B}'$ of $\mathcal{B}$ to which $\mathcal{A}$ may approximately match.

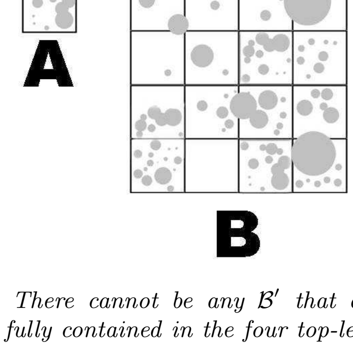


Figure 1: *There cannot be any $\mathcal{B}'$ that approximately matches $\mathcal{A}$ fully contained in the four top-left squares because $\mathcal{A}$ contains twelve disks and the squares only six.*

Once selected the set of geometric parameters to be used, in the second stage of the quadtree construction, we will traverse $\mathcal{Q}_{\mathcal{B}}$ and associate to each node the selected geometric parameters. We will also compute them for the whole of $\mathcal{A}$. The computational cost of adding the geometric information to $\mathcal{Q}_{\mathcal{B}}$ depends on the parameters that we choose. In the case of the "number of disks" and "list of radii" parameters we can easily keep track of them while we build the quadtree, so no additional cost is needed. Adding other parameters will indeed need extra computational time but will also make the discarding of zones more effective. The balance between this two factors will be an important part of our future work.

### 3.2 Candidate zones determination

We must face the problem of determining all the candidate zones where squares of size $s$ that cover a subset of $\mathcal{B}$ which is parameter compatible with $\mathcal{A}$ can be located. The subdivision induced by the nodes of size $s$ of $\mathcal{Q}_{\mathcal{B}}$ corresponds to a grid of squares of size $s$ superimposed to set $\mathcal{B}$. If we bear in mind that we are trying to place a certain square in a grid of squares of the same size, it is easy to see that the only three ways to place one of our squares respect to this grid correspond to the relative position of one of the square's vertices. This will yield three different kinds of candidate zones associated to one, two or four nodes (see Figure 2). The subsets $\mathcal{B}'$ that we are looking for may lie anywhere inside those zones.

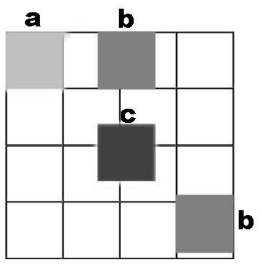


Figure 2: *Position of the candidate zones in the grid.*

#### 3.2.1 Search algorithm

Due to lack of space, we only provide a very brief overview of an algorithm that traverses the quadtree $\mathcal{Q}_{\mathcal{B}}$ searching for the collection $\mathcal{C}$ of candidate zones. The hierarchical decomposition of $\mathcal{B}$ provided by $\mathcal{Q}_{\mathcal{B}}$ makes possible to begin searching at the whole of $\mathcal{B}$ and later continue the search only in those zones where, according to the selected geometric parameters, it is really necessary. The algorithm searches recursively in all the quadrants considering also those zones that can be built using parts of more than one of them. The zones taken into account through all the search will continue to decrease their size, until they reach $s$, following the algorithm's descent of the quadtree. Consequently, early discards made on behalf of the geometric parameters will rule out of the search bigger subsets of $\mathcal{B}$ than later ones.

Given that two or four nodes defining a candidate zone need not be in the same branch of $\mathcal{Q}_{\mathcal{B}}$, at some points we will need to be exploring two or four branches simultaneously. This will force us to have three separate search functions, depending on the type of candidate zones we are looking for, and to calculate the geometric information associated to those zones that do not correspond exactly to nodes in the quadtree. This calculations are of constant cost in the case of the parameters number of disks and list of radii. The main search function seeks for candidate zones formed by only one node and the other two seek for zones formed by two or four nodes.

In the worst case all possible zones are considered to be candidate zones and the total number of nodes of $\mathcal{Q}_{\mathcal{B}} \in O(m)$, so the number of candidates zones, $c = |\mathcal{C}|$, is in $O(m)$. Considering a "perfect" behavior of the geometric pruning technique and that the quadtree is descended from the root (sized $t$) down to a level whose nodes have size $s$, $t - s < m$, then the cost of the search algorithm is $O(c(t-s)) \in O(m^2)$.

Summarizing, the computational cost of our algorithm in the case where in the two steps only the "number of disks" and "list of radii" parameters are used is $O(\max\{m \log m, c(t-s)\})$. In practice we expect the factor $(t-s)$ to be close to constant achieving computational times close to quasi-linear in $m$.

## 4 NDM problem solving algorithm

Now we have a **NDM** problem where we can expect the sets involved, $\mathcal{A}, \mathcal{S} \in \mathcal{C}$ ($n = |\mathcal{A}| \leq n' = |\mathcal{S}| \leq m$), to be "similar" in numbers of disks and in the aspects of their shape described by the geometric parameters. We present an algorithm to solve this **NDM** problem, that adapts the best currently existing algorithms for solving the **NM** problem [3, 1] and takes advantage of the compressed quadtree that we have already built and is implementable. Our approach will consist of two parts called "enumeration" and "testing".

### 4.1 Enumeration

Generating every possible rigid motion that brings set $\mathcal{A}$ onto a subset of $\mathcal{S}$ is infeasible due to the continuous nature of movement. We will partition the set of all rigid motions in equivalence classes in order to make their handling possible.

For $b \in \mathbb{R}^2$, let $(b)^\epsilon$ denote the circle of radius $\epsilon$ centered at point $b$. Let $\mathcal{S}^\epsilon$ denote the set $\{(b)^\epsilon | D(b,s) \in \mathcal{S}\}$. Consider the arrangement $\mathcal{A}(\mathcal{S}^\epsilon)$ induced by the circles in $\mathcal{S}^\epsilon$. Two rigid motions $\tau$ and $\tau'$ will be considered equivalent if for all disks $D(a,r) \in \mathcal{A}$, $\tau(a)$ and $\tau'(a)$ lie in the same cell of $\mathcal{A}(\mathcal{S}^\epsilon)$. We generate a solution in each equivalence class, when it exists, and its corresponding representative motion using the techniques presented in [1]. A simple geometric argument shows that if there exists any rigid motion $\tau$ that solves our **NDM** problem then there exists another rigid motion $\tau'$, that belongs to the equivalence class of $\tau$, that also does it and such that we can find two pairs of disks $D(a_i, r_i), D(a_j, r_j) \in \mathcal{A}$ and $D(b_k, s_k), D(b_l, s_l) \in \mathcal{S}$, $r_i = s_k$ and $r_j = s_l$, with $\tau'(a_i) \in (b_k)^\epsilon$ and $\tau'(a_j) \in (b_l)^\epsilon$. We check this property for all quadruples $i, j, k, l$.

Mapping $a_i, a_j$ onto the boundaries of $(b_k)^\epsilon$, $(b_l)^\epsilon$ respectively in general leaves one degree of freedom which is parametrized by the angle $\phi \in [0, 2\pi)$ between the vector $|a_i - b_k|$ and a horizontal line. Considering any other disk $D(a_h, r_h) \in \mathcal{A}$, $h \neq i, j$ for all possible values of $\phi$, the center of that disk will trace an algebraic curve of degree 6 $\sigma_{ijklh}$ so that for every value of $\phi$ there exists a rigid motion $\tau_\phi$ holding $\tau_\phi(a_i) \in (b_k)^\epsilon$, $\tau_\phi(a_j) \in (b_l)^\epsilon$ and $\tau_\phi(a_h) = \sigma_{ijklh}(\phi)$. For every remaining disk $D(b_p, s_p)$ in $\mathcal{S}$ with $s_p = r_h$, we compute the intersections between $(b_p)^\epsilon$ and $\sigma_{ijklh}(\phi)$ which contains at most 12 points. For parameter $\phi$, this yields a maximum of 6 intervals contained in $I = [0, 2\pi[$ where the image of $\tau_\phi(a_h)$ belongs to $(b_p)^\epsilon$. We will call this set $I_{p,h}$ following the notations in [1]). Notice for all the values $\phi \in I_{p,h}$ we may approximately match both disks. We repeat the process for each possible pair $D(a_h, r_h), D(b_p, s_p)$ and consider the sorted endpoints, called *critical events*, of all the intervals $I_{p,h}$. Notice that the number of critical events is $O(nn')$. Subsequently, any $\phi \in [0, 2\pi[$ that is not one of those endpoints belongs to a certain number of $I_{p,h}$'s and $\phi$ corresponds to a certain rigid motion $\tau_\phi$ that brings the disks in all the pairs $D(a_h, r_h), D(b_p, s_p)$ near enough to be matched. The subdivision of $[0, 2\pi[$ consisting in all the maximal subintervals that do not have any endpoints of any $I_{p,h}$ in their interior stands for the partition of the set of rigid motions that we were looking for.

## 4.2 Testing

We move parameter $\phi$ along the resulting subdivision of $[0, 2\pi[$. Every time a critical event is reached, we test the sets $\tau_\phi(\mathcal{A})$ and $\mathcal{S}$ for matching. Whenever the testing part determines a matching of cardinality $n$ we annotate the corresponding $\tau_\phi$ and proceed. Following the techniques presented in [3], in order to update the matching, we need to find a single augmenting path using a layered graph.

When searching for augmenting paths we will need to perform efficiently the two following operations: a) **neighbor**$(D(\mathcal{T}), q)$: for a query point $q$ in a data structure $D(\mathcal{T})$ that represents a point set $\mathcal{T}$, return a point in $\mathcal{T}$ whose distance to $q$ is at most $\epsilon$ or $\emptyset$ if no such element exists. b) **delete**$(D(\mathcal{T}), s)$: Delete point $s$ from $D(\mathcal{T})$. For our implementation we will use the *skip quadtree*, a data structure that combines the best features of a quadtree and a skip list [4]. The cost of building a skip quadtree for any set $\mathcal{T} \subseteq \mathcal{S}$ is in $O(n' \log n')$. This computational cost is, in the worst case when $n' = m$, the same obtained in [3]. **Neighbor** operation is used to get all the points in our skip quadtree holding the condition previously stated, and combined with the delete operation to prevent refinding points. This corresponds to a range searching operation in the skip quadtree followed by a set of deletions. The asymptotic computational cost of the **deletion** of a point in a skip quadtree is $O(\log n')$. The range searching can be approximated in $O(\delta^{-1} \log n' + u)$ time where $u$ is the size of the output in the approximate range query case, for constant $\delta > 0$ [4]. The approximate range searching outputs some "false" neighbor points that can be detected in $O(1)$ time. For the asymptotic cost of the neighbor operation, we observe that the smaller $\delta$ is, the bigger the computational time for the approximate range searching but the smaller the number of false neighbors. This leaves the cost of the **neighbor** operation between $O(\frac{\delta^{-1} \log n'}{u} + 1)$ amortized cost, when $\delta$ allows no false neighbors, and $O(n')$ when $\delta$ is arbitrarily big. The first case meets the cost in [3] and the second one is a little worse and meets the costs in [1]. We believe this last case to be unrealistic and expect an overall of our algorithm performance similar to [3] using simpler data structures. We denote $t(n')$ an upper bound on the time of performing **neighbor** operation in $\mathcal{S}$'s skip quadtree. This yields a computational cost of $O(nt(n'))$ for finding an augmenting path.

## 4.3 Computational Cost

During the enumeration part, in the worst case $O(n^2 n'^2)$ quadruples of disks are considered. For each quadruple, we work with $O(nn')$ pairs of disks, obtaining $O(nn')$ critical events. Summed over all quadruples the total number of critical events encountered in the course of the algorithm is $O(n^3 n'^3)$. Finally for the testing part, since we spent $O(nt(n'))$ time at each critical event for finding an augmenting path, the total time of the algorithm sums to $O(n^4 n'^3 t(n'))$.

When all candidate zones $\mathcal{S} \in \mathcal{C}$ are considered, the total cost is $\sum_{\mathcal{S} \in \mathcal{C}} O(n^4 n'^3 t(n'))$.

## References

[1] H. Alt, K. Mehlhorn, H. Wagener and E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete & Computational Geometry*, 3:237–256, 1988.

[2] S. Aluru and F.E. Sevilgen. Dynamic compressed hyperoctrees with application to the N-body problem. *Proc. 19th Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, LNCS 1738:21-33, 1999.

[3] A. Efrat, A. Itai and M.J. Katz. Geometry helps in Bottleneck Matching and related problems. *Algorithmica*, 31:1–28, 2001.

[4] D. Eppstein, M.T. Goodrich, and J.Z. Sun. The skip quadtree: a simple dynamic data structure for multidimensional data. *21st ACM Symp. on Comp. Geom.*, 296–305, 2005.

# So Much Data, So Little Time

Bernard Chazelle

Department of Computer Science, Princeton University
35 Olden Street, Princeton, NJ 08540-5233, USA
chazelle@cs.princeton.edu

Motivated by the need to cope with floods of data, algorithm design is undergoing profound changes. Self-improvement, online property-preserving filtering, uncertainty, sublinearity and property testing are all parts of the story. I will discuss some of these exciting developments with an emphasis on its geometric aspects.

# Restricted Mesh Simplification Using Edge Contractions

Mattias Andersson [*] Joachim Gudmundsson [†] Christos Levcopoulos[‡]

**Abstract**

We consider the problem of simplifying a triangle mesh using edge contractions, under the restriction that the resulting vertices must be a subset of the input set. That is, contraction of an edge must be made onto one of its adjacent vertices. In order to maintain a high number of contractible edges under this restriction, a small modification of the mesh around the edge to be contracted is allowed. Such a contraction is denoted a 2-step contraction. Given $m$ "important" points or edges it is shown that a simplification hierarchy of size $O(n)$ and depth $O(\log n/m)$ may be constructed in $O(n)$ time. Further, for many edges not even 2-step contractions may be enough, and thus, the concept is generalized to $k$-step contractions.

## 1 Introduction

In computer graphics objects are commonly represented using triangle meshes. One important problem regarding these meshes is how to efficiently simplify them, while maintaining a good approximation of the original mesh. As an example, scanners often produce information redundant meshes containing millions of points and triangles. Further, often the simplication should be performed in several rounds, such that a level-of-detail hierarchy is constructed. One application of such a hierarchy is that an appropriate level may be chosen depending on viewing distance, as finer details tend to be unnecessary as the distance increases. Other applications include progressive transmission and efficient storing. It is common to represent the level-of-detail hierarchy as a directed, acyclic and hierarchical graph, where each level in the graph corresponds to a level in the level-of-detail hierarchy, and where each node in the graph corresponds to a triangle, in the natural way. The first, top-most, level in the graph corresponds to the input mesh. When a contraction is made two triangles disappear, and one or more triangles are affected in such a way that their appearance change. In the graph this is represented with edges between disappearing triangles at some level $i$, and the affected triangles at level $i + 1$. Such a graph will simply be denoted a *hierarchical graph*, and the efficiency of a simplification algorithm is directly related to the size [3, 11] and depth of the hierarchical graph it may produce. Simplification algorithms constructing hierarchies of size $O(n)$ and depth $O(\log n)$ have been presented for several problem variants [1, 2, 4, 10]. Mesh simplification is generally regarded as a mature field (see [5, 8] for surveys), consisting of several suggested methods and problem variants. In this paper the method of iteratively contracting edges [1, 6, 7, 9] is considered, where contractions are made such that no crossing edges result from the contraction. This method is examined under the restriction that the set of output points is required to be a subset of the input points, i.e., contraction of an edge must be done onto one of its adjacent vertices. In order to maintain a high number of contractible edges a small modification of the mesh around an edge to be contracted is allowed. This method is denoted a 2-step contraction, and it is shown that a hierarchical graph of size $O(n)$ and depth $O(\log n/m)$ may be produced under the restriction that several "important" points or edges may not be contracted. Further, in order to enable contraction of edges that are not even 2-step contractible the concept is generalized to $k$-step contractions. We show that $k$ can be bounded by either $deg(v) - 4$ of a vertex $v$ to be contracted, or by the number of concave corners on the hull of $v$ (see Section 2 for definition).

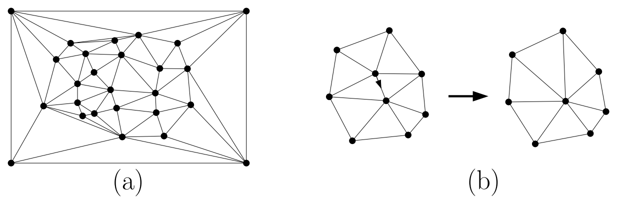


Figure 1: (a) A planar triangulation with a rectangular outer hull is given as input. (b) Illustrating an edge contraction (merge-operation).

## 2 Contracting in $k$ steps

As input we are given a planar triangulation $T$. We can assume that the outer hull of $T$ is a rectangle, as illustrated in figure 1a.

[*]Department of Computer Science, Lund University, mattias@cs.lth.se
[†]National ICT Australia Ltd., IMAGEN Program, Joachim.Gudmundsson@nicta.com.au
[‡]Department of Computer Science, Lund University, christos@cs.lth.se

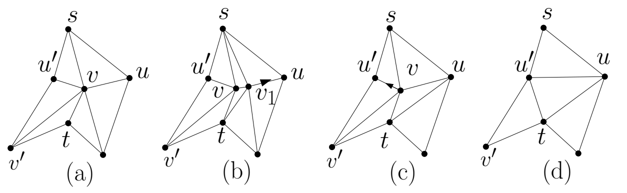


Figure 2: Illustrating a 2-step contraction of a degree 6 node $v$.

The aim is to simplify $T$ by iteratively performing edge contractions, as shown in Fig. 1b, where an edge $(u, v)$ can be contracted such that $u$ is moved to $v$, or $v$ is moved to $u$. A problem that often occurs during edge contractions of triangulations is that the resulting triangulation might not be planar. An edge contraction is said to be *valid* if the resulting triangulation still is planar. In this paper we consider the problems of defining valid contractions and computing valid contractions. Below some basic operations and notations are defined:

Consider a vertex $v$ and assume the degree of $v$ is $d$, and let $u_1, \ldots, u_d$ be the vertices adjacent to $v$ in clockwise order around $v$. The *hull* of $v$, denoted $\mathcal{H}(v)$, is the cycle described by the edges $(u_i, u_{(i+1) \bmod d})$ for $1 \leq i \leq d$. We need some additional definitions. A *split* operation splits a vertex $v$ of degree $d$ into two vertices $v$ and $v_1$ connected by an edge such that the resulting triangulation is planar, $v$ has degree $d'$ and $v_1$ has degree $d - d' + 4$, as illustrated in Figure 2a-b. Next, a *merge* operation contracts one vertex $u$ onto another vertex $u'$, both connected by an edge in $T$, into one vertex $u'$, as illustrated in Figure 2c-d. Let $d(u')$ denote the degree of $u'$ before the contraction. After the contraction $u'$ has degree $d(u) + d(u') - 3$. And, finally, a *split-and-merge* operation on vertices $v$ and $u$ first performs a split operation of $v$, followed by a merge operation on $v_1$ and $u$. The split-and-merge operation is said to be *valid* if the triangulation is planar at each step of the operation. Note that an edge contraction is obtained by a single merge operation.

Next we define 1-step contractible using the merge concept, and we then generalize this concept into $k$-step contractible. If two vertices $u$ and $v$, connected by an edge in the triangulation $T$ can be merged into a new vertex $w$ placed at $u$ such that the contracted triangulation still is planar, then $v$ is said to be *1-step contractible* (at $u$). A vertex $v$ is said to be *$k$-step contractible* if and only if one can perform at most $k - 1$ valid split-and-merge operations followed by a 1-step contraction of $v$.

With regards to guaranteeing a hierarchical simplification graph of small size and depth, mainly 2-step contractions will be considered. Figure 2 shows a vertex $v$ that is 2-step contractible since a valid split-and-

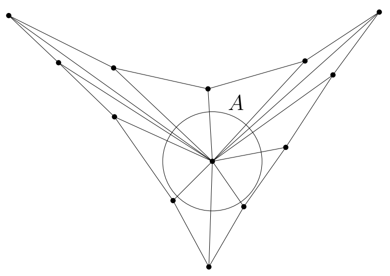


Figure 3: No edge in region $A$ can be contracted, unless using $k$-step contractions.

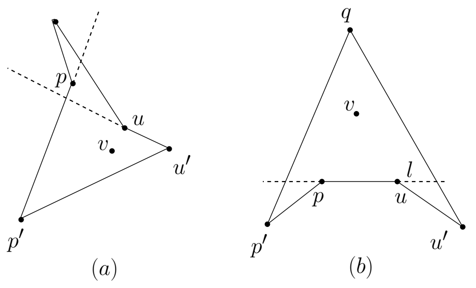


Figure 4: The two cases of Theorem 1.

merge operation is followed by a 1-step contraction of $v$. However, to be able to change the level of details for models where the user is allowed to perform the standard operations - rotate, zoom and so on, we would ideally perform edge contractions in small specified areas. For example, in configurations like the one shown in Figure 3, if one wants to perform $k$-step contractions which change or contract only edges inside the area A, then $k$ has to be at least proportional to the number of edges inside A (divided by some constant).

Thus, the generalized concept of a $k$-step contraction is needed, and below (Theorem 1 and 3) upper bounds on $k$ are shown.

**Theorem 1** *Any vertex $v$, not on the hull of $T$, with degree at most $m$ is $k$-step contractible, where $k = \max\{1, m - 4\}$.*

**Proof.** The theorem is proven by induction on the degree of $v$.
*Base case:* Vertices of degree at most four can easily be 1-step contracted. We thus assume that $v$ has degree five, which immediately implies that $\mathcal{H}(v)$ contains five points. Consider the interior of $\mathcal{H}(v)$. If there exists a corner $v'$ of $\mathcal{H}(v)$ which can see all other corners of $\mathcal{H}(v)$ then $v$ is 1-step contractible at $v'$, and thus, the theorem holds. Next, since $\mathcal{H}(v)$ has at least three convex corners, $\mathcal{H}(v)$ has at most two concave corners. If $\mathcal{H}(v)$ has only one concave corner $u$ then this corner must see all the vertices of $\mathcal{H}(v)$ and, hence, $v$ is 1-step contractible to $u$. If $\mathcal{H}(v)$ has two concave corners we have two cases, as shown in

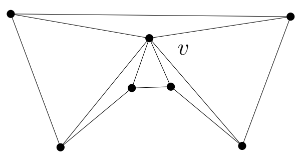


Figure 5: Example of a vertex $v$ that is not 1-step contractible.

Figure 4. Note that edges between $v$ and $\mathcal{H}(v)$ are not included in order to avoid cluttering of the figure. In the first case, Figure 4a, the concave corner points $p$ and $u$ are not incident, while in the second, Figure 4b, they are.

**First case** First note that $p$ and $u$ must lie inside the triangle defined by the three convex corners in $\mathcal{H}(v)$, and also that $p$ and $u$ always see each other. There exist two incident convex corner points $p'$ and $u'$, such that $p'$ is incident on $p$ and $u'$ is incident on $u$. It is straightforward to see that $p$ sees all points of $\mathcal{H}(v)$ if the edge $(p, p')$ does not cross the line-extension of the segment $(u, u')$, as $p$ then can see $u'$. The same holds for $u$, the edge $(u, u')$ and the line extension of $(p, p')$. However, both cases can not occur simultaneously as this implies that the edges $(u, u')$ and $(p, p')$ must cross. Thus, either $p$ or $u$ can see all of $\mathcal{H}(v)$.

**Second case** Consider the one convex corner point $q$ not incident on either $p$ or $u$. Since $q$ is connected to both $p'$ and $u'$, $q$ will see all corners of $\mathcal{H}(v)$ if and only if $q$ sees both $p$ and $u$. Next, consider the line-extension $l$ of the segment $(p, u)$. Since $p$ and $u$ are concave corners $p'$ and $u'$ must lie on the same side of $l$. Further, $q$ must connect to $p'$ and $u'$ such that $p'$ and $u'$ form convex angles and $p$ and $u$ form concave angles. This means that $q$ must lie on the opposite side of $p$ and $u$, with regards to $l$, which immediately implies that $q$ can see both $p$ and $u$.

*Induction hypothesis:* Assume that the theorem holds for all vertices of degree at most $m - 1$.
*Induction step:* Assume that $v$ has degree $m$. There exists a point $u$ on the hull of $v$ that can see at least four consecutive vertices of the hull including itself. Denote these vertices $u_1, \ldots, u_4$. Split $v$ into $v_1$ and $v$ such that $v_1$ is connected to $u_1, \ldots, u_4$ and $v$. Now, $v_1$ can be 1-step contracted at $u$. The degree of $v$ is now $m-1$, thus applying the induction hypothesis on $v$ proves the theorem. □

Note that if $v$ has degree 6 then it might not be 1-step contractible as shown in Figure 5.

**Corollary 2** *At least two edges in $T$ are* 1*-step contractible.*

**Proof.** Follows from Theorem 1 and the fact that any planar graph has total degree at most $6n-12$ (Euler's theorem). Details omitted. □

Note that if only few edges are 1-step contractible then almost all vertices in $T$ must have degree 6. The bound stated in Corollary 2 is probably very conservative. If the number of 1-step contractible edges is small that implies that almost all vertices of $T$ have degree 6. However, we have not been able to construct any examples where almost all vertices have degree 6 while simultaneously being not 1-step contractible. Next we present an alternative bound on $k$. As only concave corners restrict visibility, intuitively it should be easier to contract a vertex with few concave corners on its hull. The following theorem can be shown.

**Theorem 3** *Every vertex $v$, not on the hull of $T$, with at most $c$ concave vertices on its hull is $k$-step contractible, where $k = c$.*

**Proof.** Proof omitted. □

## 3 The number of $k$-contractible edges

Allowing $k$-step contractions increases the flexibility of simplification since it allows a greater fraction of the edges to be contracted. In this section a lower bound on the number of $k$-contractible edges is given, where Theorem 1 and the fact that the total degree is bounded is used.

**Observation 1** *At least $(\frac{k-1}{k+2})n$ vertices are $k$-step contractible, for any $k \geq 2$.*

**Proof.** Let $L$ be the set of interior vertices of $T$ that are $k$-step contractible. From Theorem 1 we know that $L$ at least consists of all vertices of degree at most $d = k+4$. Let $N$ be the remaining set of interior vertices. Recall that the sum of the degrees over all vertices is at most $6n-12$. This implies that the size of $L$ is minimized when all vertices in $N$ has degree $(d+1)$ and the vertices in $L$ and the four vertices on the hull has degree 3. We have the following equation:

$$4 \cdot 3 + (d+1)|N| + 3|L| = 6n - 12 \text{ where } |N| + |L| = n - 4.$$

As a result it holds that $|L| \geq \left(\frac{d-5}{d-2}\right)n \geq \left(\frac{k-1}{k+2}\right)n$. □

## 4 The hierarchical graph

In this section we show that using 2-step contractions we can achieve a hierarchical graph, as defined in the introduction, of size $O(n)$ and depth $O(\log n/m)$, given $m$ *important* points or edges that may not be contracted. In order to do this several edges must be simultaneously contracted in each *round*, that is, at

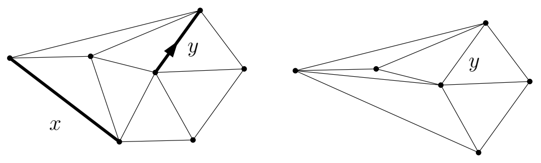


Figure 6: Initially $x$ and $y$ are contractible, but after $x$ has been contracted $y$ is no longer contractible.

each level of the graph. Next, note that a previously valid 1-step contractible edge might become invalid after other edges have been contracted, as shown in Figure 6. In order to avoid this problem, for the purpose of finding simultaneously contractable edges, we consider independent edges. Let $S_2'$ be the set of 2-step contractible vertices of degree at most six. Combining Theorem 1 and Observation 1 it is straightforward to see that $|S_2'| \geq \frac{n}{4}$. Since a vertex in $S_2'$ has at most six neighbors we can choose at least $\frac{n}{4 \cdot 7} = \frac{n}{28}$ vertices from $S_2'$ such that none of these chosen vertices has a neighbor from $S_2'$. Thus, there exists a constant fraction $\gamma$ of independent 2-step contractible vertices, and the following theorem can be shown.

**Theorem 4** *Given $m$ important points $S'' \subset S$ in a triangulation $T$ one can perform $O(\log n/m)$ rounds of 2-step contractions to obtain a triangulation $T'$ of a point set $S'$ with complexity $O(m)$ such that $S'' \subseteq S' \subset S$.*

**Proof.** Let $n_i$ denote the number of vertices before round $i$ and consider an arbitrary constant $\delta < \gamma$. Perform rounds until $m \geq \delta n_i$, that is until the resulting point set $S'$ have complexity $O(m)$. This is possible, since as long as $m \leq \delta n_i$, there are at least $\gamma n_i - \delta n_i = (\gamma - \delta) n_i$ 2-step contractible vertices remaining, containing no important point. Thus, $T'$ can be obtained using at most $O(\log_{\frac{1}{\gamma-\delta}} n - \log_{\frac{1}{\gamma-\delta}} m) = O(\log n - \log m) = O(\log n/m)$ rounds of contractions. □

**Corollary 5** *Using rounds of 2-step contractions a hierachical graph of size $O(n)$ and depth $O(\log n/m)$, given $m$ important points, may be produced in $O(n)$ time.*

**Proof.** Note that the above theorem immediately enables the construction of hierarchical graph of depth $O(\log n/m)$. Next, consider the size. Note that the number of nodes in the hierarchical graph is $O(n)$ and only 2-step contractible vertices of degree at most six are used during the rounds of contractions. This means that at most four triangles are affected by a contraction, which implies that each node in the hierarchical graph has at most four incident edges. Thus, the hierarchical graph has size $O(n)$

Next, consider the time complexity of creating the hierarchical graph. Note that the Theorem 4 was shown using only 2-step contractible edges of constant degree (at most six). Thus, in each round $i$ the set of $\gamma n_i$ independent 2-step contractible edges can be found in $O(n_i)$ time. This means, since $n_i \leq n(\gamma)^{i-1}$, that the total running time is $O(n + n\gamma + n(\gamma)^2 + \ldots + n(\gamma)^{O(\log n/m)}) = O(n(\gamma + \gamma^2 + \ldots + \gamma^{O(\log n/m)}) = O(n)$. □

Finally, note that the above results also holds for $m$ important edges (or $m$ edges and vertices, in total), since each important edge restricts possible contraction for only a constant (two) number of vertices.

## References

[1] S. Cheng, T. Dey and S. Poon. Hierarchy of Surface Models and Irreducible Triangulations. Computational Geometry Theory and Applications (CGTA), 27:135-150, 2004.

[2] M. de Berg, K. T. G. Magillo. On Levels of Detail in Terrains. In Proc. *ACM Symposium on Computational Geometry*, pp.26-27, 1995.

[3] L. De Floriani, P. Magillo and E. Puppo. Building and Traversing a Surface at Variable Resolution. In Proc. *IEEE Visualization'97*, pp.103-110, 1997.

[4] C. A. Duncan, M. T. Goodrich and S. G. Kobourov. Planarity-Preserving Clustering and Embedding for Large Planar Graphs. In Proc. *Graph Drawing '99*, pp.186-196, 1999.

[5] M. Garland. Multiresolution Modelling: Survey and Future Opportunities. Eurographics '99, State of the Art Report (STAR). http://graphics.cs.uiuc.edu/ garland/papers.html

[6] S. Gumhold, P. Borodin and R. Klein. Intersection Free Simplification In Proc. *4th Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics*, pp 11-16, 2003

[7] P. Heckbert and M. Garland. Surface Simplification Using Quadric Error Metrics. In Proc. *SIGGRAPH'97*, pp 209-216, 1997

[8] P. Heckbert and M. Garland. Survey of Polygonal Surface Simplification Algorithms. Multiresolution Surface Modelling Course, SIGGRAPH'97 http://graphics.cs.uiuc.edu/ garland/papers.html

[9] H. Hoppe. Progressive Meshes. In Proc. *SIGGRAPH'96*, pp. 99-108, 1996

[10] D. G. Kirkpatrick. Optimal Search in Planar Subdivisions. SIAM Journal of Computing, 12:28-35, 1983.

[11] J. C. Xia, J. El-Sana and A. Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In Proc. *IEEE Visualization'96*, pp.327-334, 1996.

# Guaranteed-Quality Anisotropic Mesh Generation for Domains with Curves

Yusuke Yokosuka[*] Keiko Imai[†]

## Abstract

Anisotropic mesh generation is important for interpolation and numerical modeling. Recently, Labelle and Shewchuk proposed a two dimensional guaranteed-quality anisotropic mesh generation algorithm called Voronoi refinement. This algorithm treats only domains with straight lines as inputs. However, in many applications, input domains have many curves and the exact representation of curves is needed for efficient numerical modeling. In this paper, we extend the Voronoi refinement and propose it as a guaranteed-quality anisotropic mesh generation algorithm for domains with curves. Some experimental results are also shown.

## 1 Introduction

Mesh generation is used in interpolation including computer graphics, and numerical modeling including the finite element method. It has been shown that anisotropic meshes where the elements are elongated along specified directions are well suited for interpolation and numerical modeling [6]. In this paper, we consider two dimensional anisotropic mesh generation.

In anisotropic mesh generation, many heuristic solutions have been proposed [1, 2, 4, 7]. These algorithms work for all kinds of domains. However, meshes generated by heuristics are not unique and have no guaranteed property. Recently, Labelle and Shewchuk [3] have proposed an anisotropic mesh generation algorithm which guarantees that high quality meshes are generatedD This high quality means that there are no poor-quality triangles in the mesh. The algorithm can be applied to the cases where the input domain is a planar straight line graph (PSLG). However, it cannot be used for input domains with curves. Therefore, we aim to extend this algorithm to treat domains with curves.

There is an approach to handle domains with curves. If we first approximate an input curve with segments, the Voronoi refinement algorithm proposed by Labelle and Shewchuk can be used for the segments. The new points inserted by the algorithm are not on the original curves but on the approximate segments. It follows that the obtained mesh does not precisely approximate the boundary of the input domain. To overcome this, there is a way to divide the curve into smaller segments. If we use this method, the number of triangles in the mesh is undesirably large. From the above discussion, we find that where new points are to be inserted has to be decided during the execution of the algorithm.

## 2 Anisotropic Mesh

In this section, we first explain the relationship between a metric tensor and anisotropic mesh generation. Next, we define anisotropic Voronoi diagrams and anisotropic Delaunay triangulations. These definitions are introduced by Labelle and Shewchuk [3].

Consider a domain $\Omega \subseteq \mathbb{R}^d$. Suppose that for any point $p \in \Omega$, there is a *metric tensor* $M_p$. $M_p$ is given as a symmetric positive definite matrix, and is used to measure length and angles from $p$. The distance between $q_1 \in \mathbb{R}^d$ and $q_2 \in \mathbb{R}^d$ as measured by $p$ is defined as

$$d_p(q_1, q_2) = \sqrt{(q_1 - q_2)^{\mathrm{T}} M_p (q_1 - q_2)}.$$

Let $d_p(q) = d_p(p, q)$. In the same way, the angle $\angle_p q_1 q_2 q_3$ $(q_1, q_2, q_3 \in \mathbb{R}^d)$ as measured by $p$ is defined as

$$\angle_p q_1 q_2 q_3 = \arccos \frac{(q_1 - q_2)^{\mathrm{T}} M_p (q_3 - q_2)}{d_p(q_1, q_2) d_p(q_3, q_2)}.$$

Given a metric tensor $M_p$ of a point $p$, define a *deformation tensor* $F_p$ to be any matrix such that $F_p^{\mathrm{T}} F_p = M_p$ and $\det(F_p) > 0$. The *relative distortion* between $p$ and $q$ is defined as $\tau(p, q) = \tau(q, p) = \max\{||F_q F_p^{-1}||_2,\ ||F_p F_q^{-1}||_2\}$.

In anisotropic mesh generation, the metric tensors show the directions and size of scaling at each point in the domain. Triangles are elongated according to the metric tensors. Consider any point $p$ in a triangle $t$. If $t$ is equilateral as measured by $p$, $t$ is elongated as to the metric tensor at $p$ in physical space. Therefore, in anisotropic mesh generation, it is necessary that each triangle is equilateral as measured by any point within itself.

*Department of Information and System Engineering, Graduate School of Science and Engineering, Chuo University, yoyu@imai-lab.ise.chuo-u.ac.jp

†Department of Information and System Engineering, Chuo University, imai@ise.chuo-u.ac.jp

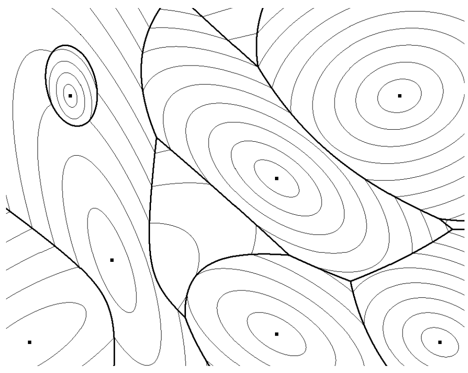


Figure 1: An anisotropic Voronoi diagram.

We briefly explain anisotropic Voronoi diagrams as proposed by Labelle and Shewchuk [3]. Let $V$ be a set of points called sites. The *Voronoi cell* of a site $v \in V$ is

$$\mathrm{Vor}(v) = \{p \in \mathbb{R}^d \mid d_v(p) \le d_w(p), \forall w \in V\}.$$

Any subset of sites $W \subseteq V$ define a Voronoi face $\mathrm{Vor}(W) = \bigcap_{w \in W} \mathrm{Vor}(w)$, which is a set of points equally close to the sites in $W$ but no closer to any other. Every site of $W$ is said to *own* $\mathrm{Vor}(W)$. The *anisotropic Voronoi diagram* of $V$ is the arrangement of the non-empty faces $\{\mathrm{Vor}(W) \mid W \subseteq V, W \neq \emptyset, \mathrm{Vor}(W) \neq \emptyset\}$. Figure 1 shows an example of an anisotropic Voronoi diagram. 0-faces and 1-faces in the Voronoi diagram are called *Voronoi vertices* and *Voronoi arcs*, respectively.

The dual of an ordinary Voronoi diagram is the Delaunay triangulation. But the dual of an anisotropic Voronoi diagram is not generally a triangulation. Here, we describe the condition where the dual of the anisotropic Voronoi diagram is a triangulation. The *wedge* between two sites $v$, $w$ is defined as

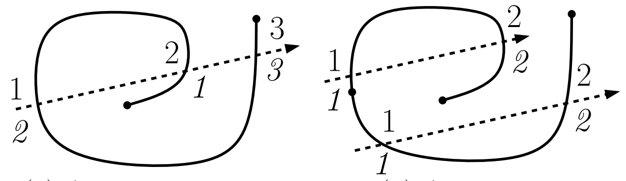
$$\begin{aligned}\mathrm{wedge}(v, w) \quad = \quad & \{q \in \mathbb{R}^d \mid (q - v)^{\mathrm{T}} M_v (w - v) > 0 \\ & \text{and } (q - w)^{\mathrm{T}} M_w (v - w) > 0\}.\end{aligned}$$

A Voronoi $k$-face $f \in \mathrm{Vor}(W)$ $(0 \le k < d,\ W \subseteq V)$ is said to be *wedged* if for any pair of sites $v_1$, $v_2 \in W$ $(v_1 \neq v_2)$, any point $q \in f$ is inside the wedge$(v_1, v_2)$.

We know the following property [3]. Let $V$ be a set of sites in a general position and let $D$ be the anisotropic Voronoi diagram of $V$. If all the Voronoi arcs and vertices in $D$ are wedged, the dual of $D$ is a triangulation. This triangulation is called the *anisotropic Delaunay triangulation*.

## 3 Voronoi refinement for Domains with Curves

The Voronoi refinement algorithm can be applied to polygonal domains with straight line segments. We want to extend the algorithm for curve-bounded domains with internal curves.



(a) A counter example (b) An example

Figure 2: A linear order.

First, we precisely describe the conditions of the input domains. The input in our algorithm is a Planar Regular Curve Graph $X$ and a metric tensor field $M$. The definition of a Planar Regular Curve Graph is as follows.

**Definition 1** *A* Planar Regular Curve Graph *(PRCG) is a set of sites and regular curves in a plane that satisfies three conditions:*

1. *For any curve contained in a PRCG, two endpoints of the curve are sites in the PRCG.*
2. *The site is an endpoint of a curve, or a point that is not on a curve.*
3. *Curves are permitted to intersect only at their endpoints.*

These conditions are equal to the conditions of a Planar Straight Line Graph (PSLG), which is a set of sites and line segments. If a set of curves is that of segments, a PRCG is identical with a PSLG. Additionally, we assume that any pair of curves has no two common endpoints. Moreover, we suppose that the collinear points on any curve occur in a linear order (Figure 2(b)), and for any two curves the two convex hulls are disjoint with the exception of shared endpoints. Even if these additional conditions are not satisfied in a PRCG, we can make changes to satisfy them in the preprocessing.

Let $\Omega \subset \mathbb{R}^2$ be a finite domain in which we want to generate a triangulation. We assume that curves and points in the interior of $\Omega$ consist of a PRCG and the boundary of $\Omega$ is also a PRCG. In total, the PRCG is designated as $X(\Omega)$. The domain $\Omega$ that satisfies this condition is called a PRCG domain.

Our algorithm generates a quality anisotropic mesh so that the following condition is satisfied: Each angle in every triangle $t$ in the mesh is larger than or equal to $\theta_{\mathrm{bound}}$ as measured by any point in $t$. In this paper, a triangle that doesn't satisfy this condition is called a poor-quality triangle.

We define some concepts to give an outline of our algorithm for the generation of a quality anisotropic mesh. If a Voronoi cell $\mathrm{Vor}(w)$ of a site $w$ contains a curve $c$ that does not contain $w$, $c$ is said to be encroached upon $w$ (Figure 3). If $c$ is encroached, split it by inserting a site $z$ in $c \cap \mathrm{Vor}(w)$. At this time,

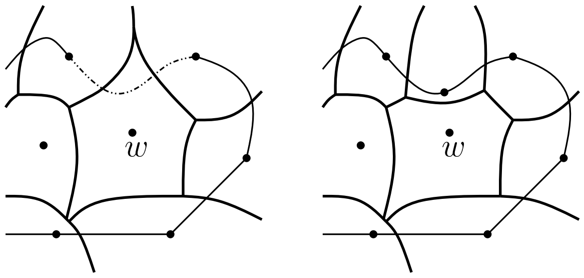


Figure 3: A curve encroached upon a site $w$ is split.

if possible, the curve $c$ is split so that $d_a(z) = d_b(z)$ where $a$ and $b$ are the endpoints of $c$. We consider the case that the point $z$ such that $d_a(z) = d_b(z)$ does not lie in $\mathrm{Vor}(w)$ of an encroaching site $w$. In this case, let $z'$ be the point that is in $c \cap \mathrm{Vor}(w)$ and as close to the point $z$ as possible, and $z'$ is inserted as a new site. This operation is called a "split".

**Definition 2** *A point $q$ in domain $\Omega$ is called a* violator *if $q$ satisfies either one of the next two conditions.*

- *$q$ lies on a Voronoi arc $\mathrm{Vor}(\{v, w\})$ $(v, w \in X)$ that is not wedged, and $q$ doesn't lie in $\mathrm{wedge}(v, w)$. In this case, $q$ is called a wedge violator.*
- *$q$ is a Voronoi vertex whose dual is an inverted or a poor-quality triangle. $q$ is called a poor violator.*

**Outline of our algorithm**
Step 1: Construct the anisotropic Voronoi diagram $D$ of sites in $X$. For each encroached curve $c$, split the curve $c$. After this, every curve is contained in the Voronoi cells of its endpoints.
Step 2: Select any violator $q$. If $q$ does not encroach, then $q$ is inserted as a new site. Otherwise, $q$ is not inserted and $c$ is split. After that, update the anisotropic Voronoi diagram.

In our algorithm, $\sin\theta_{\mathrm{bound}}$ is strictly less than 1/4 ($\arcsin(1/4) \approx 14.4°$) to guarantee termination of our algorithm. The detail of the proof is described in the next section.

## 4 Termination of the algorithm

In this section, we prove that our algorithm terminates and guarantees good quality. Most of the proof follows the same approach as in [3]. We give only a short summary of the proof here, further details about it are given in the full paper.

We assume that each angle of any two curves in the input PRCG $X$ is more than 60°. Under this assumption, we can prove that our algorithm terminates for $\theta_{\mathrm{bound}} < \arcsin(1/4)$. We will show that the algorithm does not bring segments with shorter distances

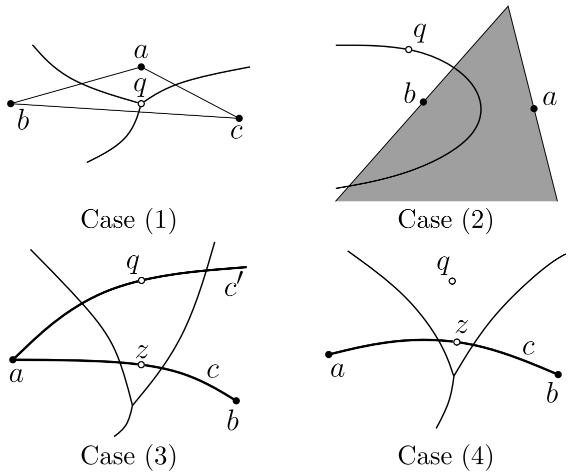


Figure 4: Inserting a new site.

than that of any segment existing in each step. In the proof, we use the Euclidean distance because we can prove the same fact in the same way even where we use a metric tensor.

There are four cases in which a new site is inserted during the execution of the algorithm (Figure 4); (1) The new site is a poor violator; (2) It is a wedge violator; (3) A site on a curve $c'$ encroaches another curve $c$ and a point on $c$ is inserted; (4) The algorithm tries to insert a violator $q$ whose $\mathrm{Vor}(q)$ encroaches another curve, and inserts a new site on the curve instead of $q$.

First, we consider the case of (1). Let $q$ be a poor violator. In this case, the dual triangle of $q$ has an angle that is less than $\theta_{\mathrm{bound}}$. $l$ denotes the shortest edge of the triangle. We can show that the length of the new segment that appears by inserting $q$ is more than $Bl$, where $B = 1/(2\sin\theta_{\mathrm{bound}})$. In the case of (2), with further investigation into details, we also have it that the length of the new segment that appears by inserting $q$ is more than $Bl$.

Next, we examine the case of (3). In this case, the site $q$ on a curve $c'$ encroaches another curve $c$. The situation that $c$ and $c'$ are disjoint is of no matter in the proof of the termination. Therefore, we assume that $c$ and $c'$ have a common endpoint $a$. Let $b$ be the other endpoint of $c$. The new site $z$ is chosen as the nearest point in $\mathrm{Vor}(q) \cap c$ from the Voronoi edge determined by $a$ and $b$. In the triangle $\triangle qaz$, the shortest edge among the newly created edges is $qz$ because $z$ is in $\mathrm{Vor}(q)$. Therefore, $\overline{qz} < \overline{az}$ and this means $\angle zqa > \angle qaz$. It is shown that $\angle qaz$ is the smallest in the triangle $\triangle qaz$ from our assumption that $\angle qaz > 60°$. Therefore, we get $\overline{qz} \geq \overline{qa}$ and the new segment is longer than the existing segments.

We consider the last case (4) is which a violator $q$ encroaches a curve $c$, and $c$ is split. Note that $q$ is not inserted in this case. The algorithm inserts the point $z$, which is the nearest point in $\mathrm{Vor}(q) \cap c$ from

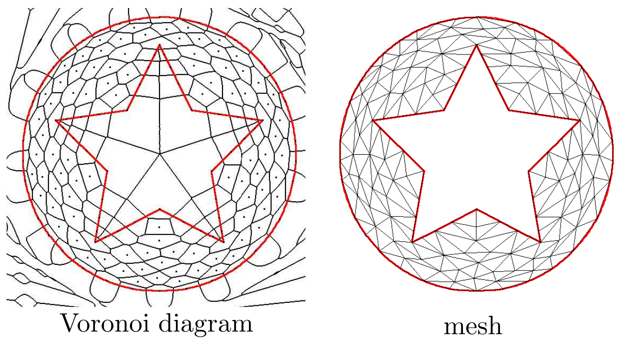


Figure 5: Input with straight lines and a circle.

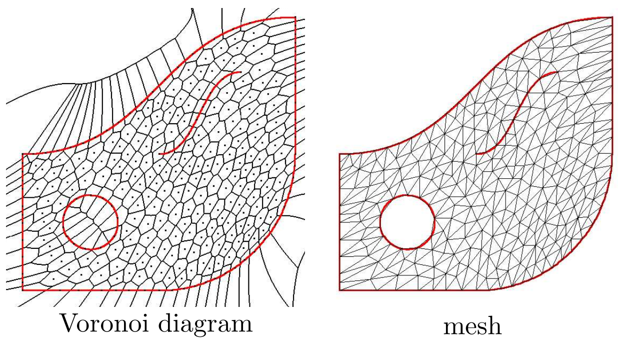


Figure 6: Input with straight lines, circles, and Bezier curves.

the Voronoi edge determined by $a$ and $b$, where $a$ and $b$ are endpoints of $c$. Without loss of generality, we assume that $q$ is nearer to $b$ than $a$. In the triangle $\triangle qbz$, we can get that $\overline{qz} + \overline{bz} > \overline{qb}$ from the triangle inequality. Moreover, $\overline{qz} < \overline{bz}$ because $z$ is in $\mathrm{Vor}(q)$. Summarizing the above discussion, we have $\overline{bz} > \overline{qb}/2$. $q$ is a violator and the distance between any other point and $q$ is greater than $Bl$. If $B > 2$, the distance from the other site to $q$ is not shorter than the length of any existing segment. This condition is satisfied when $\theta_{\mathrm{bound}} < \arcsin(1/4)$.

From the above discussion, the algorithm does not create a shorter segment than the existing segments in the case of $\theta_{\mathrm{bound}} < \arcsin(1/4)$. Let $l$ be the shortest length among the segments created during the algorithm. We draw a circle whose center is a site with radius $l/2$. There are only a finite number of such circles in $\Omega$. From this fact, we have that our algorithm terminates. When there is no violator, the algorithm also terminates. Therefore, in the obtained mesh, there is no angle that is smaller than $\theta_{\mathrm{bound}}$.

## 5 Experimental Results

In this section, we show some experimental results. We generate anisotropic meshes for 10 domains with curves by using the algorithm we described in Section 3. We assign $\theta_{\mathrm{bound}} = 14.4°$ to guarantee that our algorithm terminates. Some of these results are shown in Figure 5 and 6. Moreover, our algorithm often terminates in practice even if the specified angle is greater than $\arcsin(1/4)$.

## 6 Conclusion

In this paper, we proposed a guaranteed-quality anisotropic mesh generation algorithm for domains with curves. This algorithm has as its basis a Voronoi refinement algorithm [3]. This proposed algorithm generates an anisotropic mesh in which no triangle has an angle smaller than 14.4°, as measured by any point in the triangle. We also gave some experimental results. The results showed that the proposed algorithm does indeed generate guaranteed-quality anisotropic meshes.

There are three proposed future works. The upper bound of the specified angle $\theta_{\mathrm{bound}}$ in the original Voronoi refinement algorithm is 20.7°. However, our upper bound is 14.4°. Therefore, one future work is to improve the upper bound of the specified angle $\theta_{\mathrm{bound}}$. The second will be to take away the assumption of the lower bound of input angles. In the Delaunay refinement algorithm, Pav and Walkington's approach [5] accepts inputs without a lower bound for the input angles. But as yet there is no Voronoi refinement algorithm with such a property. The third work will be to improve our method for three dimensional anisotropic mesh generation.

## References

[1] F. J. Bossen, P. S. Heckbert, A Pliant Method for Anisotropic Mesh Generation. *Fifth International Meshing Roundtable*, 63–74, 1996.

[2] P. George, H. Borouchaki, Delaunay Triangulation and Meshing: Application to Finite Elements. Hermes, Paris, 1998.

[3] F. Labelle, J. R. Shewchuk, Anisotropic Voronoi Diagrams and Guaranteed-Quality Anisotropic Mesh Generation. *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, 191–200, 2003.

[4] X. Li, S. Teng, A. Üngör, Biting Ellipses to Generate Anisotropic Mesh. *Eighth International Meshing Roundtable*, 97–108, 1999.

[5] S. Pav, N. Walkington, Delaunay Refinement by Corner Lopping. *Fourteenth International Meshing Roundtable*, 165–182, 2005.

[6] J. R. Shewchuk, What Is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. *Eleventh International Meshing Roundtable*, 115–126, 2002.

[7] K. Shimada, A. Yamada, T. Itoh, Anisotropic Triangulation of Parametric Surfaces via Close Packing of Ellipsoids. *International Journal of Computational Geometry and Applications*, 10(4), 400–424, 2000.

# Modifying Delaunay Refined Two-Dimensional Triangular Meshes

Narcís Coll, Marité Guerrieri and J.Antoni Sellarès [*]

## Abstract

We propose algorithms to modify a mesh of a PSLG through out the interactive addition/deletion of elements to/from the PSLG, keeping the quality of the mesh all along the process. Our algorithms achieve quality by deleting, moving or inserting Steiner points.

## 1 Introduction

There exist many works on the generation of quality meshes. *Delaunay refinement mesh generation* algorithms have taken place in this frame of investigations [6, 7, 5, 4]. Different kind of domains can be meshed, but a Planar Straight Line Graph (PSLG) is the main input to 2D refining algorithms. In all these works modification of the initial PSLG implies a regeneration of the whole mesh. These mesh generation algorithms do not allow us to modify the initial PSLG incrementally.

In keeping quality of a mesh two objectives are pursued. First, get *skinny* triangles, triangles without required quality, out of the mesh. Second, force segments of the PSLG into the mesh. Both goals are achieved by the addition of Steiner points, points that do not belong to the original mesh. In current Delaunay refinement algorithms, two kinds of Steiner points deal with the former goal, namely, circumcenters and off-centers. The later objective is carried out by the addition of midpoints on constrained segments to insert.

All these algorithms guarantee that the length of the edges of the triangulation are greater than the *minimum local feature size* ($lfs_{\mathsf{min}}$) of the PSLG. This $lfs_{\mathsf{min}}$ is the shortest distance between two nonincident elements (points or segments) of the input PSLG.

Applications where a combination of dynamic modifications of a mesh and numerical methods like the Finite Element Method (FEM) is required have motivated this research. Apart from the quality requirement, these applications expected additional features to be provided by the algorithms, namely:

**Progressively:** The interactive modification of the initial PSLG are obtained without the regeneration of the whole mesh.

**Locality:** The changes applied to the mesh do not imply a propagation of these modifications to the whole mesh.

**Optimality:** The Steiner points added as a result of the modification of the mesh should be as few as possible.

Modification of quality meshes under insertion or deletion of PSLG elements required incremental algorithms. In order to obtain a new refined mesh adding the minimum number of Steiner points we have to consider that the current mesh is treated as an input PSLG by existing algorithms. Consequently, the lower bound of lengths of the edges of the new mesh will not be the $lfs_{\mathsf{min}}$ of the modified PSLG. This bound will be the $lfs_{\mathsf{min}}$ of the union of the current mesh and the modified PSLG.

Possible solutions to the excessive insertion of points could be the movement or deletion of Steiner points, or a different algorithm that split constrained segments. The movement of points, also, could resolve the problem of the generation of small triangles produced by a degradation in the distribution of points in successive updates of the mesh. Movement of points could assure better $lfs_{\mathsf{min}}$ bounds. Figure 1 shows the insertion of four consecutive segments before and after deleting Steiner points. In Figure 1(b) deleted Steiner points are shown to facilitate the understanding of the process.

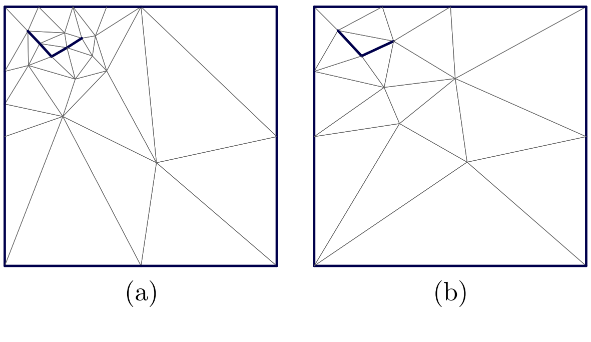


Figure 1: Insertion of four consecutive segments. Triangulation before and after deletion of Steiner points.

## 2 Quality zones

The main idea behind our proposed algorithms is that the quality of a mesh can be improved by means of the movement of Steiner points belonging to skinny triangles. This solution relies on the fact that it is possible to define a zone for a given Steiner vertex

[*]Departament d'Informàtica i Matemàtica Aplicada, Universitat de Girona, {coll,mariteg,sellares}@ima.udg.es.
Work partially supported by grant TIN 2004-08065-C02-02.

where it can be placed ensuring that the quality is preserved.

The *star*, $\mathcal{S}_q$, of a vertex $q$ of a mesh consists of all the triangles that contain $q$. All edges of triangles in $\mathcal{S}_q$ that are disjoint from $q$ form the *link*, $\mathcal{L}_q$, of $q$. The *quality zone* $\mathcal{Z}_{ab,t,\alpha}$ of an edge $ab$ of a triangle $t$ for an angle $\alpha$, that controls the quality of the triangles, is defined by the domain of points $p$ external to the circumcircle of the triangle adjacent to $t$ by $ab$ and satisfying $\widehat{apb} \geq \alpha$, $\widehat{pab} \geq \alpha$ and $\widehat{abp} \geq \alpha$. These conditions assure that the triangle $abp$ is Delaunay and non-skinny. See Figure 2 for an example of this defined zone. The *Quality zone* for a given vertex $q$ is defined by $\mathcal{Q}_{q,\alpha} = \bigcap \mathcal{Z}_{e,t_e,\alpha}$, $\forall e \in \mathcal{L}_q$ and $t_e \in \mathcal{S}_q$. The computation of $\mathcal{Q}_{q,\alpha}$ can be achieved by means of a sweep algorithm. Since the mean number triangles of $\mathcal{S}_q$ is bounded by six, then it can be inferred that the mean cost of the computation of the quality zone of $q$ is constant.

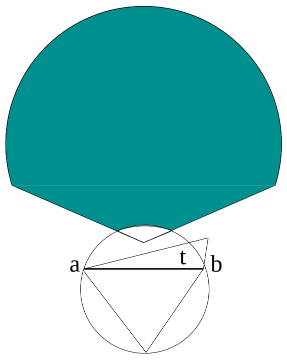


Figure 2: Quality zone of edge $ab$

## 3 Basic operations

Our main goal is the design of an algorithm that maintains a Delaunay refined mesh after the progressive insertion of elements from a PSLG. We also work under the demand of achieving local modifications of the mesh and the challenge of adding as less Steiner points as possible.

Movement and deletion of Steiner points are the *basic operations* designed to carry out these objectives. To face the appearance of skinny triangles, caused by the insertion of a new element from a PSLG, the algorithms developed try first to destroy each skinny triangle by moving or deleting its Steiner vertices, adding additional Steiner points, circumcenter or midpoints, if basic operations can not be applied. The algorithm herein described is based in a modification of an incremental Delaunay one. As it will be seen all these operations have a constant cost. The use of an incremental algorithm as well as movement and deletion of points allow us to modify a mesh in a local and progressive way, adding a lesser number of Steiner points than whether existing Delaunay refinement algorithms are used.

It has to be taken into account that points belonging to the previous mesh as well as Steiner points inserted by the modification in process are target points to be treated by our algorithm. Previous circumcenters or midpoints could be moved as a result of applying a basic operation, therefore, it is necessary to give them a new denomination. Points to be treated by our algorithm can basically belong to two main groups. One group is formed by Steiner points that are on a segment of the PSLG, whose movement is restricted to the segment. A second group is formed by those points to which their restriction of movement comes from the triangulation itself. We have named the first group *restricted vertices*, and the second group *free vertices*.

### 3.1 Moving free vertices

The key concept regarding the movement of a free vertex $p$ is to substitute this vertex for another point $q$ interior to the quality zone of $p$. If $\mathcal{Q}_{p,\alpha}$ is empty the vertex $p$ can not be moved. The problem of the election of a point $q$ that maximizes the minimum angle of $\mathcal{S}_q$ has been studied by [1] and others, but they do not include the Delaunay property as a restriction as it is required in our problem. During the sweep process used to compute the quality zone, we currently choose the midpoint of the first segment contained in the quality zone. Choosing the point into the quality zone that maximizes the smallest angle is left as a future work.

The quality zone ensures that if the vertex $p$ is placed inside it the new $\mathcal{S}_p$ is constituted by *non-skinny* triangles. The quality zone, also, produces a Delaunay triangulation of the $\mathcal{L}_p$ with respect to its exterior triangulation. But this zone does not guarantee that triangles belonging to this $\mathcal{S}_p$ fulfil Delaunay property among them. For this reason the vertex $p$ has to be deleted and reinserted in the quality zone using an Incremental Delaunay algorithm.

Consequently, the steps to determine whether a free vertex $p$ can be moved are: Determine $\mathcal{Q}_{p,\alpha}$, determine point $q$ in $\mathcal{Q}_{p,\alpha}$, delete $p$, and finally, insert $q$.

### 3.2 Deleting free vertices

Devillers in [2] proposed an algorithm to delete a point from a Delaunay triangulation. Basically, his algorithm retriangulates $\mathcal{L}_p$ by determining *Delaunay ears* using the concept of *power* of $p$. We need to obtain not just a Delaunay triangulation, but a Delaunay refined one. Consequently we verify whether the Delaunay ear is skinny or not, stopping the process when a skinny one was found. The algorithm may then be stated as:

1. Obtain a priority queue in ascending order of the power of ears from the $\mathcal{L}_p$
2. Repeat until three ears remain in the queue or a skinny triangle is found
   (a) Take the first ear from the queue and flip

```
    the diagonal to form a triangle
    (b) If the triangle is not skinny
        (i) Modify ears previous to and next to the treated ear
        (ii) Compute the power for these two new ears
        (iii) Update the priority queue
```

### 3.3 Moving restricted vertices

The movement of those restricted vertices is constrained over its correspondent subsegment. This kind of vertices can be present on a boundary subsegment or on a non boundary subsegment of a PSLG. In both cases the same algorithm to determine the movement of these vertices is applied.

The steps to determine when a restricted vertex $p$ can be moved are the followings:

```
1. Determine adjacent triangles to $p$
2. For each edge $e$ opposite to $p$
    (a) Determine quality zone, $z$
    (b) Calculate segment $i_e = z \cap s$, where $s$ is the subsegment that contains vertex $p$
3. Determine $i = \cap i_e$
4. If $i$ is not empty
    (a) Determine a point $q$ on $i$
    (b) Retriangulate: change the vertex $p$ by $q$
```

### 3.4 Deleting restricted vertices

Deletion of a restricted vertex depends on whether it belongs or not to a boundary subsegment of the PSLG. The presence of a restricted vertex on a non-boundary subsegment implies the application of the deletion algorithm explained in the section 3.2 to the two sides of the subsegment independently. In this way the point is deleted only if each side independently fulfils the point deletion verification, and then the region in each side is retriangulated individually. The same steps from the algorithm of section 3.2 can be carried out without changes, with the only extra consideration that a restricted vertex which encroached some of the polygon points can not be deleted. In case of a boundary subsegment the process detailed above is applied only to the interior side.

### 3.5 Expanding deletion of vertices

Once a vertex has been deleted from the mesh other vertices are susceptible of deletion. Each time a vertex is deleted all its adjacent Steiner vertices are added to a queue to be deleted, producing in this way several iterations. The iteration process ends when any of the vertices in the queue can not be deleted.

## 4 Modifying a 2D Delaunay refined mesh

Modification of a Delaunay refined mesh will mean to insert/delete elements to/from a PSLG. Using the basic operations we design algorithms for insertion and deletion of points, segments, polygonal lines, polygonal holes, as well as progressive polygonal lines insertions.

After the insertion or deletion of an element of the PSLG, a process of refinement is called that carries the quality through the mesh.

### 4.1 Inserting an element

To insert a point $p$ of the PSLG, we have modified the incremental Delaunay algorithm from [3]. A list of midpoints belonging to segments that could not be flipped, and a list of the generated skinny triangles is passed to the refinement process.

We used a recursive algorithm to insert a segment of the PSLG into the mesh. The process starts inserting its endpoints, then if the segment does not appear as an edge in the mesh a midpoint is added and a Delaunay process is triggered. The process of adding midpoints continues over half segments that are not yet into the triangulation.

Inserting a polygonal line into the mesh will mean to insert several segments of the PSLG one after the other. Inserting a polygon means to insert a closed polygonal line.

To obtain a polygonal hole inserted into the mesh we start inserting the polygon without any refinement, and then we delete the internal triangles.

#### 4.1.1 Inserting a polygonal line progressively

To insert a polygonal line progressively we add interactively a set of points $p_0, \cdots, p_n$. Next we describe the basic process we follow. For each point $p_i$ we insert the segment $p_{i-1}p_i$. If point $p_i$ is collinear with $p_{i-2}$ and $p_{i-1}$ then $p_{i-1}$ is considered a Steiner point and a deletion point process is started with this point.

### 4.2 Deleting an element

To delete an element of the PSLG (point, segment, polygonal line ...) the free and restricted vertices of the element are considered as Steiner points and then a deletion process is run for them.

### 4.3 Refinement process

It is a process applied after the insertion or deletion of an element of the PSLG. It receives as input two lists. The list of points to insert, initially containing only midpoints, and the list of skinny triangles to remove, produced by insertion of an element. The output of the process is a mesh with desired quality. The process maintains the two lists and finishes when the two list are empty. Priority is established on midpoints. To remove an skinny triangle, we first check its Steiner vertices for deletion, then we check its Steiner vertices for movement, and finally we add circumcenters to the list of points. This order of treatment of skinny triangles is important in order to obtain a reduction in the number of vertices. To insert a midpoint we apply our algorithm to insert a point into the mesh. To insert a circumcenter we follow rules from Ruppert's algorithm: circumcenter is not inserted if it is encroached over a constrained edge, in this case the midpoint of this edge is added to the list of points.

## 5 Results

In Figure 3 we show the insertion of a hole into the PSLG. Figure 3(a) is obtained by applying the incremental algorithm with movement and deletion of Steiner vertices and Figure 3(b) without movement or deletion of Steiner vertices. In the first case less triangles have been generated.

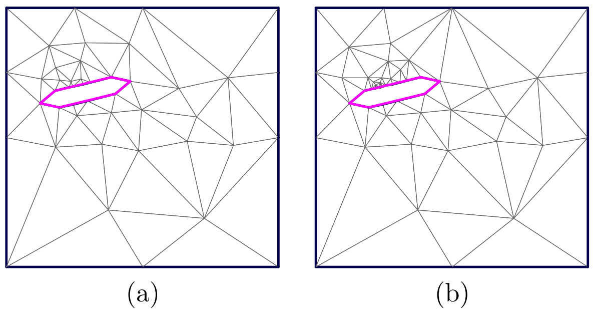


Figure 3: Insertion of a hole applying incremental algorithm.

Figure 4 shows a sequence of six steps during the insertion of a polygonal line using our algorithm. From Figure 4(a) to 4(c) a deletion operation of a vertex has been carried out. From Figure 4(d) to 4(e) a vertex has been moved. In these triangulations the Steiner points deleted are shown to facilitate the understanding of the process.

In Figure 5 the progressive insertion of the polygonal presented in the previous example is achieved using the same algorithm but without moving or deleting Steiner vertices. It can be seen that in this case we obtain more triangles.

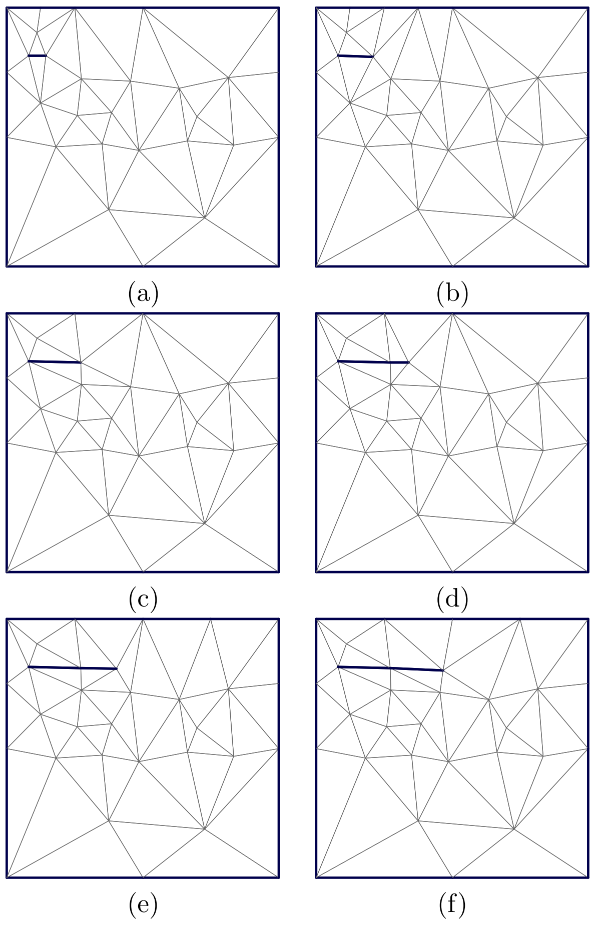


Figure 4: A sequence showing a progressive polygonal line insertion

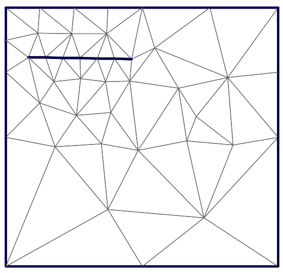


Figure 5: A progressive polygonal line insertion without moving or deleting Steiner vertices.

## References

[1] N. Amenta, M. Bern and D. Epstein. Optimal point placement for mesh smoothing. *In SODA:ACM-SIAM Symposium on Discrete ALgorithms*, 1997.

[2] O. Devillers. On deletion in Delaunay triangulation. *15th Annual ACM Symposium on Computational Geometry*, 181–188, 1999.

[3] L. Guibas, D. Knuth and M. Shair. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[4] S. Har-Peled and A. Üngör. A Time-Optimal Delaunay Refinement Algorithm in Two Dimensions. *21st Annual ACM Symposium on Computational Geometry (SoCG)*, 228–229, 2005.

[5] S.-E. Pav. Delaunay Refinement Algorithms. *Department of Mathematical Sciences - Carnegie Mellon University - PhD thesis*, 2003.

[6] J. Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18:3:548–585, 1995.

[7] J.-R. Shewchuk. Delaunay Refinement Mesh Generation. *School of Computer Science - Carnegie Mellon University - PhD thesis*, 1997.

# Mesh optimisation based on Willmore energy

Lyuba Alboul[*] Willie Brink[†] Marcos Rodrigues[‡]

**Abstract**

An algorithm for improving the quality of an initial triangulation on a fixed set of vertices is suggested. The edge flip operation is performed consecutively, aiming to minimise the discrete Willmore energy over a triangulated surface (or mesh). The Willmore energy of a surface is a function of Gaussian and mean curvature, and measures local deviation from a sphere. Virtual points are introduced in the triangulation to overcome the local invariance of Willmore energy under edge flips. Some experimental results are given.

## 1 Introduction

Computer-based modelling and visualisation has numerous applications in engineering, science and the industry. In particular, the digital capturing and reconstruction of a physical 3D object has applications in computer graphics and vision, computational geometry, reverse engineering, terrain modelling, tomography and medical imaging.

A polyhedral surface, or mesh, is a piecewise planar surface and is commonly used in computer graphics for approximating smooth surfaces. We will be concerned with triangular meshes, hereafter referred to as triangulations.

Given a set of vertices sampled from the surface of some physical 3D object, surface reconstruction is concerned with finding an "optimal" triangulation that in some sense best approximates the original surface. If the coordinates of the vertices are fixed these triangulations are said to be data-dependent.

Some examples of methods for surface reconstruction from scattered data may be found in [3, 6, 7]. These methods aim at constructing a triangulation that defines an underlying smooth surface (e.g. by means of subdivision) that somehow approximates the surface of the original 3D object in an optimal manner. Most of the reconstruction methods therefore contain an optimisation step that changes some initial triangulation on the scattered data such that the resulting triangulation induces a smooth surface that is optimal. This paper focusses on that step.

[*]Materials and Engineering Research Institute, Sheffield Hallam University, UK, L.Alboul@shu.ac.uk

[†]idem, W.Brink@shu.ac.uk (corresponding author)

[‡]idem, M.Rodrigues@shu.ac.uk

The problem that is addressed may be stated as follows: given a fixed set of points in 3D, typically acquired from the surface of a physical object by means of a 3D scanner, and some initial triangulation on these points, improve (or optimise) the local quality of this triangulation. This is generally referred to as mesh optimisation.

Some mesh optimisation algorithms based on minimising certain geometric properties of a polyhedral surface have been proposed. Examples include minimising the total area of the triangular mesh [10], minimising discrete analogues of the integral Gaussian curvature [1] and absolute mean curvature [2]. See also [5] for algorithms based on minimising these curvatures. The question of which of these algorithms produce the "best" result remains unanswered.

We propose an algorithm similar in structure to those mentioned above. However, we incorporate a geometric property known as Willmore energy. This, to our knowledge, is a new approach to mesh optimisation.

The rest of the paper is structured as follows: Section 2 briefly outlines the theory of discrete curvatures, with specific focus on the Willmore energy of a surface. Section 3 describes the mesh optimisation algorithm. Some experimental results are given in Section 4 and Section 5 concludes.

## 2 Discrete curvatures

For a smooth surface with Gaussian curvature $K$ and mean curvature $H$ the following geometric properties are of importance: the area, $\int dA$; the total Gaussian curvature, $\int K\,dA$; the total mean curvature, $\int H\,dA$; and the total Willmore energy, $\int (H^2 - K)\,dA$.

Consider a simplicial triangular surface (i.e. a triangulation) with vertex set $V$, face set $F$ and edge set $E$. Discretisations for the first three of these properties are well known for this type of surface (see for example [1, 8]).

The discrete area is simply the sum of areas of all the triangular faces.

The discrete analogue of the Gaussian curvature at a vertex $v \in V$ is defined as

$$G(v) = 2\pi - \sum_i \alpha_i,$$

where $\alpha_i$ denotes the angle at $v$ of every triangle sharing $v$. The total discrete Gaussian curvature is then given by $G = \sum_{v \in V} G(v)$.

The discrete analogue of the mean curvature at an edge $e \in E$ is defined as

$$M(e) = \theta |e|,$$

where $|e|$ denotes the length of the edge and $\theta$ the angle between the normals of the two adjacent faces. The total mean curvature is then given by $M = \sum_{e \in E} M(e)$.

Bobenko [4] recently proposed a discrete analogue of the Willmore energy for a triangulated surface. At a vertex $v$ it is defined as

$$W(v) = \sum_{e \ni v} \beta(e) - 2\pi,$$

where the sum is taken over all incident edges of $v$. For each edge $e$ the angle $\beta(e)$ is calculated as follows: let $v_i$ and $v_j$ denote the endpoints of $e$, and $v_k$ and $v_\ell$ the other two vertices of the adjacent faces, as shown in Figure 1. The value of $\beta(e)$ is then defined to be the external angle of intersection between the circumcircles of the two triangles $v_j v_i v_k$ and $v_i v_j v_\ell$.

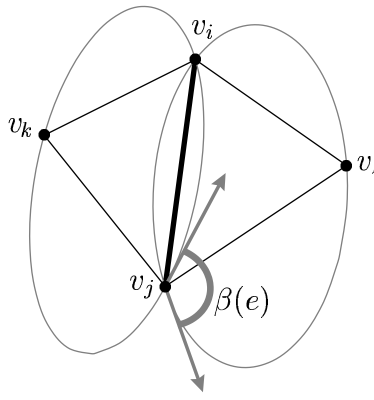


Figure 1: The angle $\beta(e)$ of an edge $e = (v_i, v_j)$.

The total discrete Willmore energy of the mesh is then given by $W = \sum_{v \in V} W(v)$.

Bobenko [4] derives some properties of this energy, most notably that $W(v) \geq 0$ and $W(v) = 0$ if and only if $v$ is convex and $v$ and all its neighbours lie on a common sphere (possibly with infinite radius, i.e. a plane).

It is therefore expected that a surface with minimum Willmore energy would be smooth and visually pleasing. This motivates the need for developing a mesh optimisation algorithm that aims at minimising Willmore energy.

## 3 Mesh optimisation

Consider a given set of vertices $V$ and some initial triangulation on these points. Our mesh optimisation algorithm attempts to minimise the discrete Willmore energy of this surface by changing the triangulation.

Following the methodology of [2], the triangulation is changed with the edge flip operation illustrated in Figure 2. A cost function is defined for a specific triangulation and edge flips that result in a decrease in this cost function are then performed successively until a minimum is reached.

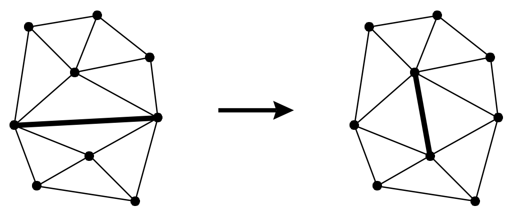


Figure 2: The edge flip operation.

The algorithm assigns to each edge a value that reflects the difference between the cost function before and after flipping the corresponding edge. We refer to this value as the cost reduction value.

The order in which the edges are flipped may be chosen by an optimisation method such as simulated annealing, but at this stage our algorithm is greedy in nature, flipping an edge that maximally reduces the cost function at every step. It is important to note that this may not always lead to a global minimum in the cost function and the algorithm may terminate at a local minimum. Techniques to escape from such a local minimum are currently under investigation. Possibile strategies include implementing multiple edge flips at each step [9].

Since the aim of our algoritm is to minimise the Willmore energy of the surface we want to define the cost function to be the total discrete Willmore energy of the current triangulation. However, as also mentioned in [4], for an edge $e$ and its flipped version $e'$, $\beta(e) = \beta(e')$. This may lead to the Willmore energy being locally invariant under flips.

Attempting to overcome this we introduce “virtual” points to the triangulation. To assign a cost reduction value to an edge $e$ we implement a simple subdivision scheme by adding a vertex $v$ in the middle of $e$ and connect it as shown in Figure 3. The total Willmore energy is calculated for this new triangulation and then compared to the total Willmore energy of the triangulation resulting from flipping $e$ to $e'$, with a point $v'$ in the middle of $e'$. The cost reduction value of $e$ is then taken to be the difference between these energies. Since in general the positions of $v$ and $v'$ would differ there would also be a difference in the Willmore energy before and after the flip.

We call the points $v$ and $v'$ virtual since they only appear in calculating the cost reduction values, not in the resulting triangulation. We refer to the Willmore energy of the triangulation with added points as the virtual Willmore energy.

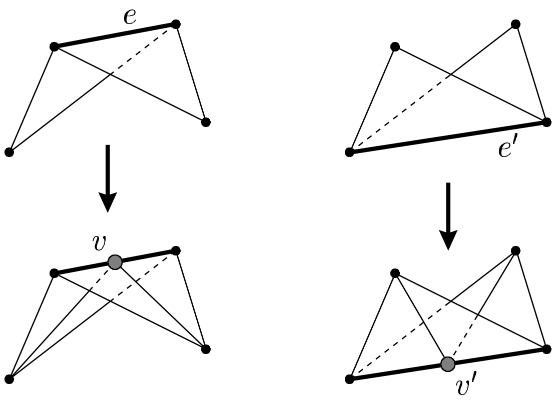


Figure 3: Adding points to the triangulation.

In one of the proofs of Bobenko [4] the combinatorics of a mesh is changed by adding points to every edge and connecting them in a similar way as depicted in Figure 3. This is done in order to render an abstract simplicial sphere inscribable and is therefore essentially quite different from what we are doing.

In the implementation of the algorithm there are some types of edges that cannot be flipped without changing the topological type of the triangulation. An edge $e$ with flipped version $e'$ should not be flipped if $e'$ is already an edge of the triangulation. For such edges we define the cost reduction function to be $-\infty$.

Note also that $\beta(e)$ is undefined for a boundary edge $e$ (i.e. an edge with only one adjacent face). It is also clear that a boundary edge cannot be flipped. Hence the cost reduction function of boundary edges are also defined as $-\infty$.

It is important to realise that the algorithm described above does not necessarily minimise the total discrete Willmore energy of the triangulated surface since this energy might remain unchanged under edge flips. The virtual Willmore energy (VWE) might be a new type of energy somehow related to the Willmore energy. It can be said that our algorithm attempts to minimise the VWE over a triangulation in the hopes of minimising the Willmore energy of the underlying smooth surface induced by the triangulation. Whether or not this would always be achieved is still under investigation although experiments do suggest that it would.

## 4 Results

This section provides some experimental results from applying our mesh optimisation algorithm on a few test models.

The data of the first model comprises of the 8 corners of a cube with 6 vertices added to the centres of each face, slightly inside the cube. Figure 4 shows on the left an initial triangulation on these vertices. The result of applying our algorithm on this triangulation is shown on the right of the figure.

What is interesting about this result is that the initial triangulation is a so-called tight triangulation [1], i.e. a triangulation on the data with minimum total absolute extrinsic curvature. Our algorithm changes this triangulation to what appears to be a triangulation with minimum total area. An algorithm that minimises total absolute curvature [2] has the exact opposite effect on this data. Exactly how these algorithms relate to each other is a topic for further study.

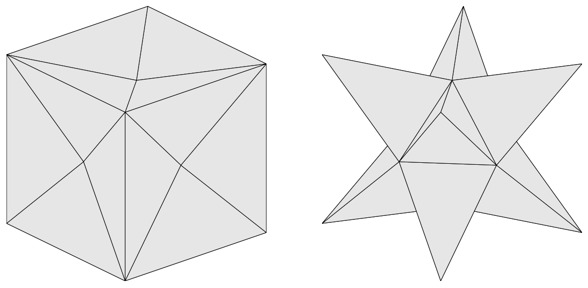


Figure 4: Test model I - a cube with 6 added vertices, initial (left) and optimised (right).

The second test model consists of points sampled on the surface of a torus. Figure 5 shows an initial triangulation on the left. On the right of the figure the result of applying our algorithm is shown.

The resulting triangulation is clearly more regular (the triangles are more or less equal in size). The triangulation is also visually smoother. This may be due to the fact that for a region of a smooth surface that closely resembles a sphere the corresponding Willmore energy is close to zero. It would seem that our algorithm attempts to extract regions in a triangulation that is spherelike.

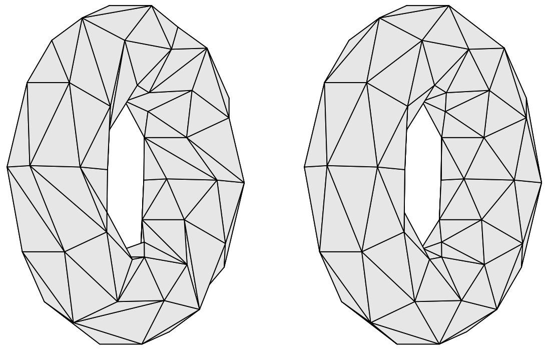


Figure 5: Test model II - points sampled on a torus, initial (left) and optimised (right).

The data for the third test model was acquired with a 3D scanner and consists of points on the surface of a human face. An initial triangulation, shown on the top left of Figure 6, was obtained by parameterising the data and applying 2D Delaunay triangulation. The result from our algorithm is shown on the top right of the figure. The figure also shows shaded versions of the top parts of these two triangulations.

There are many "vertical" edges visible in the initial triangulation. This is a parameterisation artifact and results in the underlying smooth surface to have many vertical creases (see lower left part of Figure 6). Our algorithm does seem to smooth out these creases (a result from flipping most of the vertical edges in the initial triangulation), as can be seen on the lower right of Figure 6.

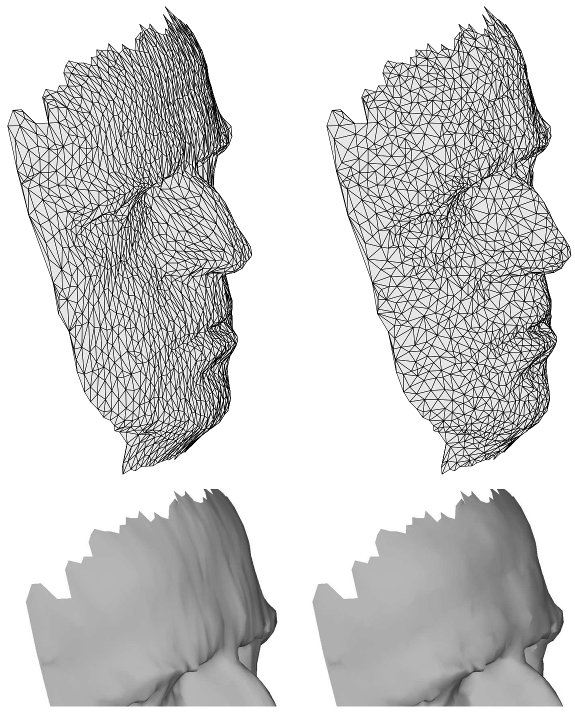


Figure 6: Test model III - points sampled on a face, initial (left) and optimised (right).

Regarding the compexity of the algorithm, consider a triangulation with $n$ edges. By keeping record of adjacency in the mesh (edges to faces) a single edge flip can be performed in $O(1)$ time. Searching through a priority queue for which edge to flip would require $O(\log n)$ time. The worst case scenario, in which every edge is flipped, would thus be $O(n \log n)$.

## 5 Conclusion and future work

We presented a new mesh optimisation algorithm. The algorithm attempts to minimise the so-called virtual Willmore energy of a triangulation by performing the edge flip operation successively.

An important area for further research is to study and develop methods for escaping from local minima in the cost function. Implementing other optimisation strategies such as simulated annealing, rather than our greedy method, might prove to be useful.

Another important issue in the algorithm is that some edge flips can result in the surface intersecting itself. Detecting these edges and avoiding such intersections is still an open problem.

Based on results obtained experimentally our algorithm seems to be promising. It should be stressed that the arguments upon which the algorithm is built are mostly heuristic in nature and a comprehensive analytical analysis is necessary.

Topics of current ongoing research also include determining under what circumstances the choice of where to position the virtual points on the edges affects the outcome, and the relationship between this algorithm and other algorithms such as minimising area or mean curvature.

The discrete analogue of Willmore energy is relatively new. It would be interesting to learn how different areas in shape modelling could benefit from this concept.

## References

[1] L. Alboul and R. van Damme. Polyhedral metrics in surface reconstructions. In: *The Mathematics of Surfaces VI*, G. Mullineux (Ed.), Clarendon Press, Oxford, 171–200, 1996.

[2] L. Alboul, G. Kloosterman, C. Traas and R. van Damme. Best data-dependent triangulations. *Journal of computational and applied mathematics*, 119:1–12, 2000.

[3] F. Bernardini and C.L. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. *Proceedings of the 9th Canadian conference of computational geometry*, 193–198, 1997.

[4] A.I. Bobenko. A conformal energy for simplicial surfaces. *Combinatorial and computational geometry*, 52:133–143, 2005.

[5] N. Dyn, K. Hormann, S.J. Kim and D. Levin. Optimizing 3D triangulations using discrete curvature anlysis. *Mathematical methods for curves and surfaces*, Oslo, 135–146, 2000.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle. Surface reconstruction from unorganized points. *ACM SIGGRAPH '92*, 71–78, 1992.

[7] K. Hormann. From scattered samples to smooth surfaces. *Proceedings of the 4th Israel-Korea bi-national conference on geometric modeling and computer graphics*, 1–5, 2003.

[8] M. Meyer, M. Desbrun, P. Schröder and A.H. Barr. Discrete differential-operators for triangulated 2-manifolds. In: *Visualization and mathematics III*, Hege and Polthier (Eds.), 35–57, 2003.

[9] A. Netchaev. Triangulations and their application in surface reconstruction. *PhD thesis*, University of Twente, The Netherlands, September 2004.

[10] J. O'Rurke. Triangulation of minimal area as 3D object models. *Proceedings of the international joint conference on AI 81*, Vancouver, 664–666, 1981.

# A New Approximation Algorithm for Labeling Weighted Points with Sliding Labels*

Thomas Erlebach† Torben Hagerup‡ Klaus Jansen§ Moritz Minzlaff¶ Alexander Wolff‖

**Abstract**

This paper presents a polynomial-time approximation algorithm for labeling some of the points in a given set of weighted points with horizontally sliding labels of unit height and given lengths to maximize the weight of the labeled points. The approach is based on a discretization, and two results are established: In general, the algorithm has an approximation factor of $2/3 - \epsilon$, for arbitrary fixed $\epsilon > 0$. If the ratio of maximal to minimal label lengths is bounded by a constant, the approximation factor becomes $1 - \epsilon$.

## 1 Introduction

*Map labeling* is the problem of placing a set of labels next to a given set of points in the plane while meeting certain conditions. Most often, the label associated with a point is of a specified rectangular shape and must be placed in the plane without rotation so that its boundary touches the point. One distinguishes *fixed-position models* and *slider models*. In fixed-position models, the labeled point must belong to a predetermined finite set of points on the boundary of the label. Slider models allow the labeled point to touch the label anywhere along a certain segment of the label's boundary. Poon et al. [3] introduce a hierarchy of fixed-position and slider models.

In this paper we consider the slider model 1SH [3] that is defined as follows. Let $P$ be a set of $n$ points in the Euclidian plane $\mathbb{R}^2$. The $x$ and $y$ coordinates of a point $p$ are denoted by $p_x$ and $p_y$, respectively. We associate with each point $p \in P$ an axes-parallel open rectangular shape $L_p$ of unit height and length $l_p \in \mathbb{R}_{>0}$, the *label* of $p$. Each point $p \in P$ has a *weight* $w_p \in \mathbb{R}_{>0}$. An instance of a 1SH-labeling problem is given by the triple $I = (P, l, w)$. A *(1SH-) labeling* of $I$ is a family $\mathcal{L} = \{r_p\}_{p \in Q}$, indexed by the elements of $Q \subseteq P$, where $r_p \in [0, l_p]$ places $L_p$ in the plane with its right edge at the $x$-coordinate $p_x + r_p$ and its lower edge at the $y$ coordinate $p_y$; see Fig. 1. For any two points $p, q \in Q$, the values of $r_p$ and $r_q$ must be such that $L_p \cap L_q = \emptyset$. Points in $Q$ are said to be *labeled*. We define $w_{\mathcal{L}} := \sum_{p \in Q} w_p$ to be the *weight* of the labeling $\mathcal{L} = \{r_p\}_{p \in Q}$. A labeling of $I$ is *optimal* if no labeling of $I$ has a larger weight. We denote the weight of an optimal labeling of $I$ by $w_{\text{opt}}(I)$.

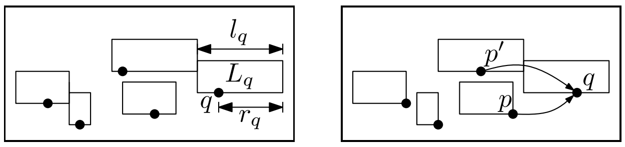


Figure 1: *Left*: A 1SH-labeling $\mathcal{L}$. *Right*: The corresponding normalization $\mathcal{L}^*$ and $G_{\mathcal{L}^*}$.

Poon et al. [3] show that finding an optimal 1SH-labeling is NP-hard even if all points lie on a horizontal line and the weight of each point equals the length of its label. For the one-dimensional case, they give a fully polynomial-time approximation scheme, which yields an $O(n^2/\epsilon)$-time $(1/2-\epsilon)$-approximation for the two-dimensional case. Poon et al. also give a polynomial-time approximation scheme (PTAS) for unit-square labels. They raise the question of whether a PTAS exists for rectangular labels of arbitrary length and unit height. This is known to be the case for fixed-position models [1] and for sliding labels with unit weight [4].

In this paper we make a step towards settling the question. We present a new approximation algorithm for 1SH-labeling. If the ratio of maximal to minimal label lengths is bounded by a constant, our algorithm is a PTAS. In the general case our algorithm has an approximation factor of $2/3 - \epsilon$, for arbitrary fixed $\epsilon > 0$. This is an improvement over the approximation factor of $1/2 - \epsilon$ of Poon et al. [3].

In Section 2 we discretize the problem. The idea is to compute, for a given problem instance $I = (P, l, w)$, "suitable" sets of discrete label positions for each point in $P$. "Suitable" means that the weight $w_{\text{opt}}(I_{\text{fix}})$ of an optimal labeling of the resulting instance $I_{\text{fix}}$ of a fixed-position labeling problem must be close enough to $w_{\text{opt}}(I)$. This leads to a two-step approximation algorithm for the slider model 1SH:

*Work supported by grant WO 758/4-2 of the German Research Foundation (DFG).

†Department of Computer Science, University of Leicester, Leicester LE1 7RH, England. te17@mcs.le.ac.uk

‡Institut für Informatik, Universität Augsburg, D–86135 Augsburg, Germany. hagerup@informatik.uni-augsburg.de

§Institut für Informatik und Praktische Mathematik, Universität Kiel, D–24098 Kiel, Germany. kj@informatik.uni-kiel.de

¶moritz.minzlaff@stud.uni-karlsruhe.de

‖Fakultät für Informatik, Universität Karlsruhe, P.O. Box 6980, D-76128 Karlsruhe, Germany. WWW: i11www.ira.uka.de/people/awolff

First discretize the given problem instance. Then apply an algorithm, e.g., the one in [1], to the computed fixed-position problem instance. In Section 3 we consider the requirements for and the quality of the discretization.

## 2 Approximation by a Fixed-Position Model

By the *diameter* of a directed graph $G$ we mean the maximum number of nodes on any path in $G$. As a first step towards discretization, we associate with each labeling $\mathcal{L}$ a directed graph $G_{\mathcal{L}}$. If, for some constant $t \in \mathbb{N}$, $G_{\mathcal{L}}$ has a subgraph of weight at least $(1 - 1/t)w(\mathcal{L})$ and of diameter bounded by a constant $g$, then we can compute an instance $I$ of a fixed-position labeling problem such that an optimal labeling of $I$ has weight at least $(1 - 1/t)w(\mathcal{L})$. Roughly speaking, in order to compute the instance $I$, we enumerate all positions that a label can have if it is part of a chain of at most $g$ labels that succeed each other in the following sense.

**Definition 1** *Let $\mathcal{L} = \{r_p\}_{p\in Q}$ be a labeling and let $p, q \in Q$. The label $L_q$* succeeds *$L_p$ (with respect to $\mathcal{L}$), written $L_p \to L_q$, if (a) the left edge of $L_q$ touches the right edge of $L_p$ other than at a corner, (b) the vertical line supporting the left edge of $L_q$ contains no point in $Q$, and (c) $r_q \neq 0$.*

The position of each label in a chain of labels that succeed each other depends on the position of the first label in the chain. The following definition discretizes the position of the first label and thus of all its successors.

**Definition 2** *Let $\mathcal{L} = \{r_p\}_{p\in Q}$ be a labeling. For $q \in Q$, a label $L_q$ is in* normal *position (with respect to $\mathcal{L}$) if (a) $r_q = 0$, (b) the vertical line supporting the left edge of $L_q$ contains a point in $Q$, or (c) $L_q$ succeeds $L_p$ for some $p \in Q$. If all labels of $\mathcal{L}$ are in normal position, then $\mathcal{L}$ is* normal.

We can obtain a normal labeling from an arbitrary labeling by processing the labeled rectangles in the order from left to right and moving each rectangle left until it first reaches a normal position. We call this process *normalizing* the labeling. For an example see Fig. 1.

We could associate a directed graph $G = (Q, E)$ with a labeling $\{r_p\}_{p\in Q}$ by defining $(p, q) \in E$ if and only if $L_p \to L_q$. However, this is not entirely satisfactory: On the one hand, we want to bound the diameter of the graph. On the other hand, we want to normalize labelings. However, normalization may increase the diameter by too much. For this reason we also define an edge whenever there is the "possibility" that a label $L_q$ succeeds a label $L_p$, namely if a normalization after the removal of some other labels can make this happen. This gives rise to the edge $(p, q)$ in Fig. 1.

**Definition 3** *Let $\mathcal{L} = \{r_p\}_{p\in Q}$ be a labeling. The* labeling graph *$G_{\mathcal{L}} = (Q, E)$ of $\mathcal{L}$ has the edge $(p, q)$ if*

*(E1)* $p_x < q_x$,

*(E2)* $p_x + r_p > q_x - l_q$,

*(E3)* $(p_y, p_y + 1) \cap (q_y, q_y + 1) \neq \emptyset$, *and*

*(E4) there is no $p' \in Q$ such that*
$p_x + r_p \le p'_x \le q_x - l_q + r_q$.

For a label $L_q$ to succeed a label $L_p$, the point $p$ must lie to the left of $q$ (E1). Properties (E2) and (E3) ensure that $L_p$ and $L_q$ overlap if $L_q$ is shifted as far left as possible. Finally, (E4) implies that there is no point in $Q$ between the right edge of $L_p$ and the left edge of $L_q$. Note that $L_p \to L_q$ implies (E1)–(E4). We define the *weight* of a labeling graph $G_{\mathcal{L}}$ as the weight of the labeling $\mathcal{L}$. The following lemma lists properties of labeling graphs that we will need later.

**Lemma 1** *Every labeling graph is a planar directed acyclic graph. If $(p, q)$ is an edge of the graph, then this edge is the only path from $p$ to $q$.*

Now we can state precisely the condition under which we can discretize a 1SH-labeling problem such that the optimal weight of the resulting discrete instance is close enough to that of the original instance.

**Theorem 2** *Assume that for some instance $I = (P, l, w)$ of the 1SH-labeling problem, there are $g \in \mathbb{N}$ and $\alpha \in \mathbb{R}$ such that for every labeling $\mathcal{L}$ of $I$, there is a normal labeling $\mathcal{L}^*$ of $I$ with $w_{\mathcal{L}^*} \ge \alpha w_{\mathcal{L}}$ for which the diameter of $G_{\mathcal{L}^*}$ is bounded by $g$. Then, in $O(n^{g+1})$ time, we can compute for each $p \in P$ a set $M(p) \subseteq [0, l_p]$ of cardinality $O(n^g)$ such that the instance $I_{\text{fix}}$ of the fixed-position labeling problem defined by the sets $M(p)$ fulfills $w_{\text{opt}}(I_{\text{fix}}) \ge \alpha w_{\text{opt}}(I)$.*

**Proof.** For $\tau = 1, \dots, g$, we compute for each $p \in P$ a set $M(p, \tau)$ that contains all possible values of $r_p$ in a normal labeling whose longest chain of labels ending in $L_p$ contains $\tau$ labels.

$$M(p, 1) = \big(\{q_x - (p_x - l_p) \mid q \in P\} \cup \{0\}\big) \cap [0, l_p]$$

for all $p \in P$, and for $\tau = 2, \dots, g$ and for all $p \in P$,

$$M(p, \tau) = \\ \{q_x + r - (p_x - l_p) \mid q \in P, r \in M(q, \tau - 1)\} \cap [0, l_p].$$

Finally, let $M(p) = \bigcup_{\tau=1}^{g} M(p, \tau)$. Clearly $|M(p, 1)| \le n + 1$ and, by induction, $|M(p, \tau)| = O(n^\tau)$ for $\tau = 1, \dots, g$. Thus $|M(p)| = O(n^g)$ for all $p \in P$, and $M(p)$ can be computed for all $p \in P$ in $O(n^{g+1})$ time.

If $\mathcal{L}$ is an optimal labeling of $I$, then, by assumption on $g$, there is a normal labeling $\mathcal{L}^* = \{r_p^*\}_{p\in Q}$ ($Q \subseteq P$) of $I$ whose weight is at least $(1-1/t)w_{\text{opt}}(I)$ and whose labeling graph $G_{\mathcal{L}^*}$ has diameter at most $g$. By the construction above, $r_p^* \in M(p)$ for each $p \in Q$. Thus $\mathcal{L}^*$ is a labeling of the fixed-position problem instance $I_{\text{fix}}$ defined by the sets $M(p)$, so $w_{\text{opt}}(I_{\text{fix}}) \geq (1-1/t)w_{\text{opt}}(I)$. □

## 3 Simplifiable Graphs

To satisfy the prerequisites of Theorem 2, we are interested in families of labeling graphs for which, for each constant $t \in \mathbb{N}$, there is a constant $g = g(t)$ such that every labeling graph $G$ in the family is *trimmable* for $g$, i.e., contains a subgraph $G'$ with weight $w' \geq (1-1/t)w$ and diameter at most $g(t)$. As a tool we use the stronger notion of simplifiable graphs, which we now define.

**Definition 4** *Let $G = (V, E)$ be a directed graph. A* simplification *of $G$ is a function $f : V \to \mathbb{Z}$ with the following properties:*

*(S1)* $\forall (v, w) \in E: \ 0 \leq f(w) - f(v) \leq 1$.

*(S2) For all $v \in V$, there exists at most one $w \in V$ such that $(v, w) \in E$ and $f(v) = f(w)$.*

*The graph $G$ is said to be* simplifiable *by $f$ and for each $i \in \mathbb{Z}$, the set $V_f(i) = \{v \in V \mid f(v) = i\}$ is called the $i$th* level *of $G$ with respect to $f$.*

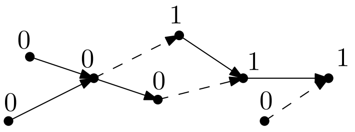


Figure 2: A graph with a simplification. Dashed arrows indicate a change of levels.

We write $V(i)$ instead of $V_f(i)$ whenever $f$ is clear from the context. Given a graph $G = (V, E)$ and a subset $V'$ of $V$, we denote by $G[V']$ the subgraph of $G$ induced by $V'$. Now we exploit property (S1) to reduce the trimming of graphs to that of levels.

**Lemma 3** *Let $G = (V, E)$ be a labeling graph with simplification $f$ and let $t \in \mathbb{N}$. Assume that for each $i \in \mathbb{Z}$, there is a subset $V'(i)$ of $V(i)$ such that $G[V'(i)]$ has weight at least $(1-1/(2t))w(G[V(i)])$ and diameter at most $g'(t)$. Then there is a subgraph $G'$ of $G$ with weight at least $(1-t)w(G)$ and diameter at most $(2t-1)g'(t)$.*

**Proof.** Consider the subgraphs

$$G_\tau := G[V \setminus \bigcup_{i\in\mathbb{Z}} V(\tau + 2ti)], \ \tau = 0, \ldots, 2t-1,$$

in which we remove every $(2t)$th level of $G$. By the pigeon-hole principle, there is a $\tau_0 \in \{0, \ldots, 2t-1\}$ such that $G_{\tau_0}$ has weight at least $(1-1/2t)w(G)$. Due to property (S1), the nodes of any path in $G_{\tau_0}$ span at most $2t-1$ distinct levels of $G$. Hence, with $V' = \bigcup_{i\in\mathbb{Z}} V'(i)$, the graph

$$G' := G[V' \setminus \bigcup_{i\in\mathbb{Z}} V'(\tau_0 + 2ti)]$$

corresponding to $G_{\tau_0}$ has the required property. □

Property (S2) implies that each node has at most one successor within its own level. Thus a level $V(i)$ is an in-forest; see Fig. 2. For $r = 0, \ldots, 2t-1$, let $V'_r(i)$ be the subset of $V(i)$ obtained by removing all nodes whose depth in the forest modulo $2t$ is $r$. By the pigeon-hole principle, the set $V'(i)$ with maximum weight among the sets $V'_r(i)$ satisfies the requirements of Lemma 3 with $g'(t) = 2t-1$.

**Theorem 4** *Every simplifiable labeling graph is trimmable for $g(t) = (2t-1)^2$.*

### 3.1 Outerplanar labeling graphs

**Lemma 5** *Let $F$ be the subgraph of a labeling graph spanned by the nodes on the boundary of one of its faces. For every edge $(v, w)$ of $F$, there are simplifications $f$ and $f'$ of $F$ with $f(w) = f(v)$ and $f'(w) = f'(v) + 1$.*

This lemma depends on the last property noted in Lemme 1. The proof is simple and can be found in [2].

**Theorem 6** *Every outerplanar labeling graph is simplifiable.*

**Proof.** A simplification can be constructed essentially as follows. Begin by choosing an arbitrary inner face of the graph. By Lemma 5, a simplification of this face exists. Then extend the simplification face by face to a simplification of the entire graph. The clue is that by outerplanarity, there is always a face that shares either exactly one node or one edge with the current subgraph. Hence extending the simplification can be done with a repeated application of Lemma 5. Tree parts of the graph can also be handled easily. We refer to [2] for a detailed proof. □

We apply this result in the context of *stabbing lines*. A set $\mathcal{S}$ of stabbing lines with respect to a labeling $\{r_p\}_{p\in Q}$ is a set of horizontal lines with the following properties: Each line has distance greater than 1 from every other line, each line intersects at least one label, and each label is intersected by exactly one line. With each stabbing line $l$ we may associate a "sub-labeling" consisting of all $r_p$ such that $L_p$ intersects $l$. For more on stabbing lines, see, e.g., [1, 2].

**Proposition 7** *If $\mathcal{L}$ is a labeling with two stabbing lines, then the labeling graph $G_{\mathcal{L}}$ is outerplanar.*

**Proof.** In a natural planar embedding of $G_{\mathcal{L}}$, each node can be reached from above or from below by a vertical ray from infinity that does not cross any edges. Therefore all nodes of $G_{\mathcal{L}}$ lie on the boundary of the outer face. □

The following corollary is an immediate consequence of the results of this section, Theorem 2 and the two-step algorithm outlined in Section 1. Its proof again relies on the pigeon-hole principle: Consider removing all labels stabbed by every third line. An optimal fixed-position labeling for $k$ stabbing lines can be computed in $A_k(\tilde{n}) = O(\tilde{n}^{2k-1})$ time, where $\tilde{n}$ is the size of the fixed-position instance [1].

**Corollary 8** *There exists an approximation algorithm with factor $2/3(1-1/t)$ for the slider model that runs in $O(n^{g(t)+1}) + A_2(n^{g(t)+1}) = O(n^{3(g(t)+1)})$ time, where $g(t) = (2t-1)^2$.*

## 3.2 Bounded Ratio of Label Lengths

The labeling graph of Fig. 3 is not simplifiable: As the numbers next to the nodes indicate, on the path from $a$ via $b$ to $c$ any simplification would need to "jump" at least two levels.

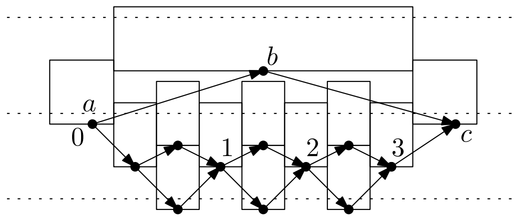


Figure 3: A labeling graph whose labeling needs three stabbing lines.

The example easily extends in such a way as to force a simplification to make arbitrarily large "jumps". However, the ratio of the maximal label length $M$ to the minimal label length $m$ would tend to infinity in the process. This gives rise to the following idea: let $\lambda \in \mathbb{N}$ and replace (S1) in the definition of a simplification by

(S1$_\lambda$) $\forall (v,w) \in E:\ 1 \le f(w) - f(v) \le \lambda$

We call a function $f$ with properties (S1$_\lambda$) and (S2) a $\lambda$-simplification and the corresponding graph $\lambda$-simplifiable. Note that $f(w) \ge f(v) + 1$ for each edge $(v,w)$ according to this definition. Therefore, each level $V(i)$ induces a subgraph of diameter 1 (consisting of isolated nodes). In the proof of Lemma 3 we can set $V'(i) = V(i)$ and have $g'(t) = 1$. If further not only one but $\lambda$ consecutive levels out of $t$ levels are removed from the graph to get $G_\tau$, the pigeon-hole principle yields the following analog of Theorem 4.

**Theorem 9** *If $G$ is a $\lambda$-simplifiable labeling graph of a labeling $\mathcal{L}$ and $t$ is an integer with $t > \lambda$, then $G$ contains a subgraph with weight at least $(1-\lambda/t)w(\mathcal{L})$ and diameter at most $g(t) = t - \lambda$.*

**Theorem 10** *Let $I$ be a problem instance with $M/m \le \rho$ for some $\rho \in \mathbb{N}$. Then the labeling graph of every labeling of $I$ is $(2\rho)$-simplifiable.*

**Proof.** Consider a labeling $\mathcal{L} = \{r_p\}_{p \in Q}$ of $I$. The idea is to divide the $x$-axis into intervals of length $m$ and to assign a node $p$ of $G_{\mathcal{L}}$ to level $i$ if the $x$-coordinate of the left edge of $L_p$ belongs to the interval $[im, (i+1)m)$. This is achieved by the following function $f : Q \to \mathbb{Z}$:

$$f(p) = \left\lfloor \frac{p_x - l_p + r_p}{m} \right\rfloor,\ p \in Q.$$

Let $(p,q)$ be an edge of the labeling graph. By properties (E1), (E2) and (E3) and the fact that $L_p$ and $L_q$ do not overlap, the left edge of $L_q$ is at least $l_p \ge m$ and at most $l_p + l_q \le 2M$ to the right of the left edge of $L_p$. Therefore $1 \le f(q) - f(p) \le 2\rho$, which shows $f$ to be a $(2\rho)$-simplification of $G_{\mathcal{L}}$. □

Plugging Theorem 10 into Theorem 9 and using a PTAS for fixed-position models [1] yields a PTAS for instances with bounded ratios of label lengths.

**Corollary 11** *For all instances with $M/m \le \rho$ for some $\rho \in \mathbb{N}$ and for all integers $t > 2\rho$ and $k \ge 1$, there exists a factor-$(1-2\rho/t)(1-1/(k+1))$ approximation algorithm that runs in $O(A_k(n^{g(t)+1}))$ time, where $g(t) = t - 2\rho$.*

## Acknowledgment

We thank an anonymous referee, whose recommendations led to greater readability.

## References

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11:209–218, 1998.

[2] M. Minzlaff. Beschriften gewichteter Punkte mit verschiebbaren Labeln. Student research report, Fakultät für Informatik, Universität Karlsruhe, March 2005. Available at `http://i11www.ira.uka.de/teaching/theses/files/studienarbeit-minzlaff-05.pdf`.

[3] S.-H. Poon, C.-S. Shin, T. Strijk, T. Uno, and A. Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.

[4] M. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.

# A polynomial-time approximation algorithm for a geometric dispersion problem

Marc Benkert* Joachim Gudmundsson† Christian Knauer‡ Esther Moet§ René van Oostrum§
Alexander Wolff*

## Abstract

We consider the problem of placing a set of disks in a region containing obstacles such that no two disks intersect. We are given a bounding polygon $P$ and a set $\mathcal{R}$ of possibly intersecting unit disks whose centers are in $P$. The task is to find a set $\mathcal{B}$ of $m$ disks of maximum radius such that no disk in $\mathcal{B}$ intersects a disk in $\mathcal{B} \cup \mathcal{R}$, where $m$ is the maximum number of unit disks that can be packed.

Baur and Fekete showed that the problem cannot be solved efficiently for radii that exceed 13/14, unless $P = NP$. In this paper we present a 2/3-approximation algorithm.

## 1 Introduction

The problem of packing objects into a bounded region is one of the classic problems in mathematics and theoretical computer science, see for example the monographs [7, 9] which are solely devoted to this problem, and the survey by Tóth [8].

In this paper we consider a problem related to packing disks into a polygonal region. As pointed out by Baur and Fekete in [1], even when the structure of the region and the objects is simple, only very little is known, see for example [4, 6]. We consider the following geometric dispersion problem:

**Problem 1** (ApproxSize) *Given a bounding polygon $P$ and a set $\mathcal{R}$ of, possibly intersecting, unit disks whose centers are in $P$, the aim is to pack $m$ non-intersecting disks of maximum radius in $P$, where $m$ is the maximal number of unit disks that can be packed in $P$.*

Note that we do not know the value of $m$ a priori. In 1985 Hochbaum and Maas gave a PTAS for the problem of packing a maximal number of unit disks in a region in their pioneering work [5]. The problem is known to be NP-complete [3]. Even though the corresponding geometric dispersion problem looks very similar much stronger inapproximability results have been shown. Baur and Fekete [1] proved hardness results for a variety of geometric dispersion problems, and their results can be modified to our setting with a bit of effort. Specifically, they show that ApproxSize cannot be solved in polynomial time for disks of radius exceeding 13/14. Furthermore, for the case when the objects are squares, Baur and Fekete gave a 2/3-approximation algorithm. However, since a square is a simpler shape and easier to pack than a disk their approach cannot be generalized to disks. The main contribution of this paper is a polynomial time 2/3-approximation algorithm. Actually, we conjecture that 2/3 is indeed the largest value for which the problem can be solved, but so far we have been unable to prove it.

ApproxSize has applications in non-photorealistic rendering system, where 3D models are to be rendered in an oil painting style, as well as in random examinations of, e.g., soil ground.

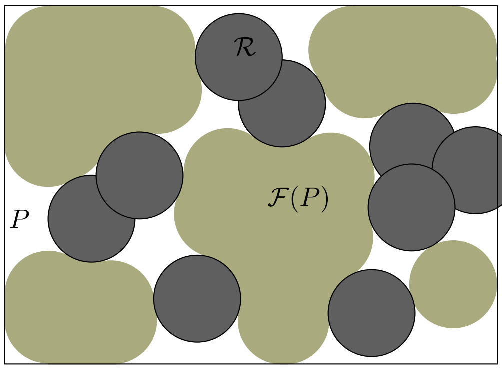


Figure 1: The freespace $\mathcal{F}(P)$ (light shaded) is the region that could be covered by a unit disk not intersecting any disk of $\mathcal{R}$.

## 2 The approximation algorithm

We will use the term $r$-disk to refer to a disk of radius $r$. The main idea of our approximation algorithm is described next, see also Algorithm 1. First compute the space $\mathcal{F}(P)$, denoted *freespace*, that could potentially be covered by a unit disk not intersecting any disk of $\mathcal{R}$. Then, apply the PTAS of Hochbaum and Maas [5] for the problem of packing unit disks in

*Faculty of Computer Sci., Karlsruhe University, Germany. Supported by grant WO 758/4-1 of the DFG. Part of this work was done while M. Benkert visited National ICT Australia, supported by a DAAD grant.
†National ICT Australia Ltd, Sydney, Australia.
‡Inst. of Computer Sci., Freie Universität Berlin, Germany.
§Dept. of Comp. Sci., Utrecht University, The Netherlands.

**Algorithm 1**: DiskPacking

```
1. Compute the freespace $\mathcal{F}(P)$.
2. Use HM's algorithm [5] to compute a set $\mathcal{B}$ of
   at least $\frac{12}{13}m$ unit disks in $\mathcal{F}(P)$.
3. Introduce a metric $d$ on the set $\mathcal{B}$ of unit disks.
4. Compute the nearest neighbor graph
   $G = (\mathcal{B}, E)$ with respect to $d$.
5. Find a sufficiently large matching in $G$.
6. For each matching pair $\{C, D\}$ of 1-disks do
7.     Place three $\frac{2}{3}$-disks in $T_{2/3}(C, D)$.
8. For each unmatched unit disk $D$ do
9.     Place one $\frac{2}{3}$-disk in $D$.
```

$\mathcal{F}(P)$. If we set $\varepsilon = 1/13$ in the PTAS this yields a set $\mathcal{B}$ of at least $12/13 \cdot m$ unit disks and it requires $O(n^{625})$ time to compute. Here, $n$ is the minimum number of unit squares that cover $P$.

Note that the approximation scheme by Hochbaum and Maas can be modified such that the algorithm is strongly polynomial with respect to the size of the input. If the number of disks that can be packed is not polynomial in the size of $P$ and $\mathcal{R}$ then there must exist a huge empty square region within $P$. This can be "cut out" and packed almost optimally by using a naïve approach. The added error obtained is bounded by $O(1/\tilde{n}^2)$ where $\tilde{n}$ is the optimal number of disks that can be packed in the square. This step can be repeated until there are no more huge empty squares.

Let $m'$ be the number of unit disks in the set $\mathcal{B}$. Starting from $\mathcal{B}$ we compute a set $\mathcal{B}_{2/3}$ of disks of radius 2/3 that has cardinality at least $13/12 \cdot m'$ which in turn yields that $\mathcal{B}_{2/3}$ contains at least $m$ disks. We obtain $\mathcal{B}_{2/3}$ by computing a sufficiently large matching in the nearest neighbor graph of $\mathcal{B}$. Then, we define a region for each matching pair such that one can insert three 2/3-disks in each region and all regions are pairwise disjoint. For each unmatched unit disk we insert one 2/3-disk, see Figure 2.

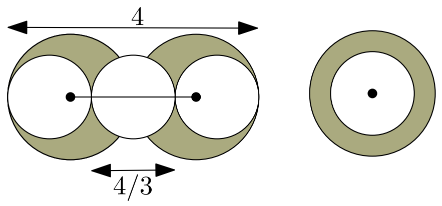


Figure 2: Packing three $\frac{2}{3}$-disks in a matching pair of unit disks (left) and one in a single disk (right).

In the next sections we describe each step of Algorithm 1 more detailed.

## 3 The freespace and a metric on it

We briefly recall the setting. We are given a set $\mathcal{R}$ of unit disks whose centers lie in a polygon $P$. The disks in $\mathcal{R}$ are allowed to intersect.

**Definition 1** *The* freespace $\mathcal{F}(P)$ *is the union of all unit disks $D$ in $P$ such that $D \cap \bigcup_{D' \in \mathcal{R}} D' = \emptyset$.*

For an example of a freespace see Figure 1. Obviously, $\mathcal{F}(P)$ can be computed in $O(n^{625})$ time. From now on we will w.l.o.g. assume that $\mathcal{F}(P)$ consists of only one connected component, since each component can be handled separately. Next, we introduce a metric for a set of non-intersecting disks in $\mathcal{F}(P)$.

**Definition 2** *Let $C$ and $D$ be two non-intersecting unit disks in $\mathcal{F}(P)$. There is a shortest movement of a unit disk from the position of $C$ to the position of $D$ that keeps entirely within $\mathcal{F}(P)$. The distance $d(C, D)$ is the length of the center-point curve $\tilde{c}(C, D)$ of this movement. We call the orbit that is induced by the transformation of the unit disk the* 1-transformation tunnel *$T(C, D)$.*

The curve $\tilde{c}(C, D)$ can consist of straight-line segments (the disk can be moved arbitrarily in the freespace without hitting any obstacles) and of arcs of radius 2 (the disk hits a disk $R \in \mathcal{R}$ on the boundary of the freespace), see Figure 3.

Next, we define a transformation for 2/3-disks. The transformation tunnels of this movement yield us the regions in which we will place the $m$ 2/3-disks.

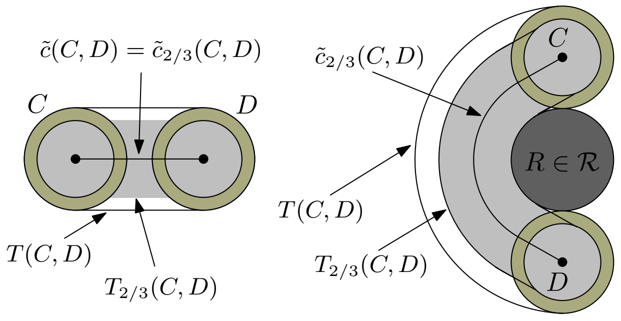


Figure 3: The minimum transformations of disks $\{C, D\}$ and $\{C', D'\}$ in $\mathcal{F}(P)$. Left: unrestricted case, right: a disk $R \in \mathcal{R}$ as obstacle.

**Definition 3** *Let $C$ and $D$ be two non-intersecting unit disks in $\mathcal{F}(P)$. Let $C_{2/3}$ and $D_{2/3}$ be the $\frac{2}{3}$-disks centered at the centers of $C$ and $D$, respectively. There is a shortest movement of a 2/3-disk from the position of $C_{2/3}$ to the position of $D_{2/3}$ that keeps entirely within $T(C, D)$. We call the orbit that is induced by the transformation of the 2/3-disk the* $\frac{2}{3}$-transformation tunnel *$T_{2/3}(C, D)$ and its center-point curve $\tilde{c}_{2/3}(C, D)$.*

## 4 The set $\mathcal{B}$ and its nearest neighbor graph

Now, the freespace $\mathcal{F}(P)$ is the region for which we compute a 12/13-approximation algorithm for the problem of packing the maximum number of unit disks. We do this by applying the PTAS of Hochbaum and Maas for $\varepsilon = 1/13$, with running time of $O(n^{625})$. Let the resulting set of unit disks be $\mathcal{B}$. By a post-processing step we can assume that $\mathcal{F}(P) \setminus \bigcup_{B \in \mathcal{B}} B$ does not offer enough space for another unit disk (inserting unit disks in a greedily manner until no more disks can be added). We need this to ensure that the nearest neighbor graph of $\mathcal{B}$ (w.r.t. $d$) is planar and of bounded degree. Using a similar argument as in the proof [2] showing that the nearest neighbor graph of a point set in the plane (w.r.t. the Euclidean metric) has degree at most 6, we can show that the degree of the nearest neighbor graph $G$ of $\mathcal{B}$ is also bounded by 6. Furthermore, $G$ is planar since no two edges in a nearest neighbor graph can intersect. Obviously, $G$ can be computed in $O(n^{625})$ time.

From now on we will call a pair $\{C, D\} \subseteq \mathcal{B}$ a *nearest pair* if $\{C, D\}$ is an edge in $G$, i.e., either $D$ is the nearest disk to $C$ (in $\mathcal{B}$) or $C$ is the nearest disk to $D$ (in $\mathcal{B}$). For every nearest pair $\{C, D\}$ we define the region $\mathcal{A}(C, D)$ to be $C \cup D \cup T_{2/3}(C, D)$. As the nearest pair $\{C, D\}$ is a potential candidate to become a matching pair, we want to ensure that we can use $\mathcal{A}(C, D)$ to pack three 2/3-disks in it such that all these packed 2/3-disks are pairwise disjoint. Thus, we have to prove: (i) three 2/3-disks fit into $\mathcal{A}(C, D)$ and (ii) for any nearest pair $\{E, F\}$ where $C, D, E$ and $F$ are pairwise disjoint, $\mathcal{A}(C, D)$ does not intersect $\mathcal{A}(E, F)$. Note that we do not have to care whether, e.g., $\mathcal{A}(C, D)$ intersects $\mathcal{A}(C, E)$ because the matching will choose at most one pair out of $\{C, D\}$ and $\{C, E\}$. Clearly, three 2/3-disks fit into $\mathcal{A}(C, D)$ since $C$ and $D$ do not intersect. Thus, (i) is fulfilled but the second part (ii) requires much more work.

We split the proof into two parts. The first part shows that $T_{2/3}(C, D)$ is not intersected by any other disk than $C$ and $D$. The second part shows that no two tunnels $T_{2/3}(C, D)$ and $T_{2/3}(E, F)$ can intersect.

We start with a technical lemma that will help to prove the first part. The notation $|p, q|$ will indicate the Euclidean distance between two points $p$ and $q$.

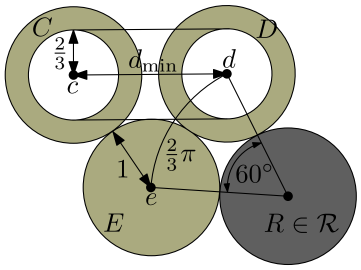


Figure 4: Illustration for Lemmas 1 and 2.

**Lemma 1** *Let $C$ and $D$ be two unit disks in $\mathcal{F}(P)$ with centers $c$ and $d$. If $|c, d|$ is less than $d_{\min} := \frac{2}{3} \cdot \sqrt{11}$, the center-point curve $\tilde{c}_{2/3}(C, D)$ is straight and each unit disk that does not intersect $C \cup D$ cannot intersect $T_{2/3}(C, D)$.*

**Proof.** We compute $d_{\min}$ as the minimum value of $|c, d|$ when a disk $E$, disjoint from $C$ and $D$, intersects $T_{2/3}(C, D)$. Clearly, $d_{\min}$ is attained if $E$ and $T_{2/3}(C, D)$ only intersect in a single point and furthermore, also $E$ and $C$ as well as $E$ and $D$ only intersect in a single point, see Figure 4. This means that $|c, e| = |d, e| = 2$, where $e$ is the center of $E$. Moreover, the Euclidean distance between $e$ and the straight-line segment $cd$ is $1 + \frac{2}{3} = \frac{5}{3}$. By Pythagoras' theorem we calculate $d_{\min} = |c, d|$ to be $\frac{2}{3}\sqrt{11}$. □

We make the following observation:

**Observation 1** *Let $C$ and $D$ be two unit disks in $\mathcal{F}(P)$ that are infinitesimally close to each other. Then $d(C, D) \leq \frac{2}{3}\pi$.*

**Proof.** For simplification we assume that $C$ and $D$ touch. The curve $\tilde{c}(C, D)$ attains its longest length if both $C$ and $D$ touch an obstacle disk that has to be overcome. In this case $\tilde{c}(C, D)$ describes an arc of radius 2 and 60°. Its length is $\frac{1}{6} \cdot 2 \cdot 2\pi = \frac{2}{3}\pi$. □

**Lemma 2** *Let $\{C, D\} \subseteq \mathcal{B}$ be a nearest pair with the center-point curve $\tilde{c}(C, D)$. Then, no disk of $\mathcal{B} \cup \mathcal{R} \setminus \{C, D\}$ intersects $T_{2/3}(C, D)$.*

**Proof.** It immediately follows from Definitions 1 and 3 that neither $T(C, D)$ nor $T_{2/3}(C, D)$ are intersected by a disk in $\mathcal{R}$. Thus, it remains to prove that apart from $C$ and $D$ no disk in $\mathcal{B}$ intersects $T_{2/3}(C, D)$.

W.l.o.g. let $C$ be the nearest disk to $D$ (in $\mathcal{R}$). The proof is done by contradiction: assume that there is a disk $E \in \mathcal{B}$ that intersects $T_{2/3}(C, D)$.

First, we move a unit disk from the position of $D$ on the center-point curve $\tilde{c}(C, D)$ to the first position in which it hits $E$, denote the disk in this position by $\overline{D}$, see Figure 4 where $D = \overline{D}$ holds. According to Observation 1, we know that $\frac{2}{3}\pi \approx 2.09$ is an upper bound on $d(\overline{D}, E)$. As $C$ is closer to $\overline{D}$ than $E$ is, the center of $C$ has to lie within a disk of radius $\frac{2}{3}\pi$ with center $\overline{d}$. However then, according to Lemma 1, $\tilde{c}_{2/3}(C, D)$ must be a straight-line because $\frac{2}{3}\pi < \frac{2}{3}\sqrt{11} \approx 2.21$ holds. Thus, also according to Lemma 1, $E$ cannot intersect $T_{2/3}(C, D)$ which yields the contradiction. □

Lemma 2 settles that no other disks apart from $C$ and $D$ intersect $T_{2/3}(C, D)$. We still have to show that any two $\frac{2}{3}$-transformation tunnels $T_{2/3}(C, D)$ and $T_{2/3}(E, F)$ do not intersect.

**Lemma 3** *Let $\{C, D\}, \{E, F\} \subseteq \mathcal{B}$ be two nearest pairs such that $C, D, E$ and $F$ are pairwise disjoint. Then $T_{2/3}(C, D) \cap T_{2/3}(E, F) = \emptyset$.*

**Proof.** The proof is by contradiction again. Obviously, we can w.l.o.g assume that $T_{2/3}(C, D)$ and $T_{2/3}(E, F)$ only intersect in a single point $p$, see Figure 5. The basic idea of the proof is to show that then, $\{C, D\}$ and $\{E, F\}$ cannot be nearest pairs at the same time. Note that at least one of the disks $\{C, D, E, F\}$ intersects the unit disk $P$ with center $p$: otherwise there would be another disk in $\mathcal{B}$ located in the space between $C, D, E$ and $F$ which would immediately contradict $\{C, D\}$ as well as $\{E, F\}$ being nearest pairs.

W.l.o.g. let $C$ be a disk that intersects $P$. We can show that $d(C, E) < d(E, F)$ holds, i.e. that $F$ is not the nearest neighbor of $E$, thus $E$ has to be the nearest neighbor of $F$ in order for $\{E, F\}$ to be a nearest pair. Under this assumption we can then show that $d(C, E) < d(C, D)$ and $d(D, F) < d(C, D)$ holds. However, this contradicts $\{C, D\}$ being a nearest pair because neither $D$ is the nearest neighbor of $C$ nor $C$ is the nearest neighbor of $D$. □

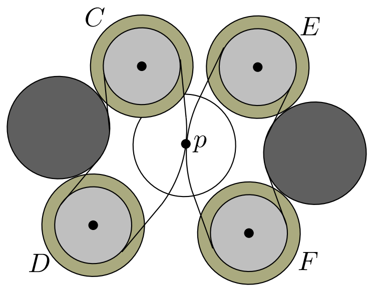


Figure 5: Illustration for the proof of Lemma 3. If $T_{2/3}(C, D)$ and $T_{2/3}(E, F)$ intersect, not both pairs $\{C, D\}$ and $\{E, F\}$ can be nearest pairs.

## 5 Placing the 2/3-disks

After computing $\mathcal{B}$ and the nearest neighbor graph $G = (\mathcal{B}, E)$, we compute a matching in $G$. Let $m' = |\mathcal{B}|$ be the number of unit disks in $\mathcal{B}$. Recall that $G$ is planar and of bounded degree 6. We show that we can find a matching in which the number of matched disks is at least $1/6 \cdot m'$. Observe that $G$ can consist of more than one connected component. We look at each connected component separately. Let $C$ be a connected component and $c$ be the number of disks that it contains. Clearly, $C$ contains a spanning tree of bounded degree 6. It is easy to see that there is a matching in $C$ that matches at least $1/6 \cdot c$ disks. Doing this for each component yields a matching in $G$ that contains at least $1/6 \cdot m'$ matched disks.

According to Lemmas 2 and 3 we can pack three 2/3-disks in $\mathcal{A}(C, D)$ for every matched pair $\{C, D\}$ such that the set of these 2/3-disks is pairwise disjoint. For each of the remaining unmatched disks $D$ we pack one 2/3-disk in $D$. Lemma 2 ensures that these disks are disjoint to the disks that have been packed for the matched pairs. Let $\mathcal{B}_{2/3}$ be the set of all disks packed as above. Its cardinality is at least $\frac{1}{6} \cdot \frac{3}{2} \cdot m' + \frac{5}{6} \cdot m' = \frac{13}{12} \cdot m'$. Since the cardinality of $\mathcal{B}$ is at least $\frac{12}{13} \cdot m$, the set $\mathcal{B}_{2/3}$ contains at least $m$ 2/3-disks and we can conclude with the following theorem:

**Theorem 4** *Algorithm 1 is a 2/3-approximation for the* ApproxSize *problem. Its running time is $O(n^{625})$.*

## 6 Conclusion

Naturally, our result is purely of theoretic interest. The bottleneck for the running time is the application of Hochbaum and Maas' PTAS. To obtain an algorithm with better running time, it seems to be unavoidable to use a completely different approach. For future work it would also be desirable to narrow the gap between the known approximation (2/3) and the inapproximability result (13/14). We conjecture that, unless $P = NP$, the lower bound of 2/3 is indeed tight.

## References

[1] C. Baur and S.P. Fekete. Approximation of Geometric Dispersion Problems. *Algorithmica*, 30:451–470, 2001.

[2] D. Eppstein and M.S. Paterson and F.F. Yao. On nearest-neighbor graphs. *Discrete & Computational Geometry*, 17(3):263-282, 1997.

[3] R.J. Fowler and M.S. Paterson and S.L. Tanimoto. Optimal packing and covering in the plane are NP-compete. *Information Processing Letters*, 12:133–137, 1981.

[4] Z. Füredi. The densest packing of equal circles into a parallel strip. *Discrete & Computational Geometry*, 6:95–106, 1991.

[5] D. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and VLSI. *J. of the ACM*, 32, 1985.

[6] C. Maranas, C. Floudas and P. Pardalos. New results in the packing of equal circles in a square. *Discrete Mathematics*, 128:187–293, 1995.

[7] C. A. Rogers. Packing and Covering. *Cambridge University Press*, 1964.

[8] G. F. Tóth. Packing and Covering. In Handbook of Discrete and Computational Geometry, 2nd edition, J. E. Goodman and J. O'Rourke, editors, CRC Press LLC, 2004.

[9] C. Zong and J. Talbot. Sphere Packings. Springer-Verlag, 1999.

# Covering a Set of Points with a Minimum Number of Lines

Magdalene Grantson Christos Levcopoulos [*]

## Abstract

We consider the minimum line covering problem: given a set $S$ of $n$ points in the plane, we want to find the smallest number $l$ of straight lines needed to cover all $n$ points in $S$. We show that this problem can be solved in $O(n \log l)$ time if $l \in O(\log^{1-\epsilon} n)$, and that this is optimal in the algebraic computation tree model (we show that the $\Omega(n \log l)$ lower bound holds for all values of $l$ up to $O(\sqrt{n})$). Furthermore, a $O(\log l)$-factor approximation can be found within the same $O(n \log l)$ time bound if $l \in O(\sqrt[4]{n})$. For the case when $l \in \Omega(\log n)$ we suggest how to improve the time complexity of the exact algorithm by a factor exponential in $l$.

## 1 Introduction

We consider the minimum line covering problem: given a set $S$ of $n$ points in the plane, we want to find the smallest number $l$ of straight lines needed to cover all $n$ points in $S$. The corresponding decision problem is: given a set $S$ of $n$ points in the plane and an integer $k$, we want to know whether it is possible to find $k$ (or fewer) straight lines that cover all $n$ points in $S$.

Langerman and Morin [7] showed that the decision problem can be solved in $O(nk + k^{2(k+1)})$ time. In this paper we show that the decision problem can be solved in $O(n \log k + (k/2.2)^{2k})$ time.

Kumar *et al.* [6] showed that the minimum line covering problem is APX-hard. That is, unless $P = NP$, there does not exist a $(1 + \epsilon)$-approximation algorithm. In their paper they pointed out that the greedy algorithm proposed by Johnson [5], which approximates the set covering problem within a factor of $O(\log n)$, is the best known approximation for the minimum line covering problem. In this paper we show that a $O(\log l)$-factor approximation for the minimum line covering problem can be obtained in time $O(n \log l + l^4 \log l)$.

We also present an algorithm that solves the line covering problem exactly in $O(n \log l + (l/2.2)^{2l})$ time. This simplifies to $O(n \log l)$ if $l \in O(\log^{1-\epsilon} n)$, and we show that this is optimal in the algebraic computation tree model. That is, we show that the $\Omega(n \log l)$ lower bound holds for all values of $l$ up to $O(\sqrt{n})$. We also suggest more asymptotic improvements for our exact algorithms when $l \in \Omega(\log n)$.

[*]Dept. of Computer Science, Lund University, Box 118, 221 Lund, Sweden {magdalene,christos}@cs.lth.se

## 2 Preliminaries

**Lemma 1** *Any set $S$ of $n$ points in the plane can be covered with at most $\lceil \frac{n}{2} \rceil$ straight lines.*

**Proof.** A simple way to show this upper bound is to pick two points at a time, to construct a line through the pair, and then to remove the pair from the set. For the special case when $n$ is odd, we can draw an arbitrary line through the last point. The time complexity of this algorithm is obviously $O(n)$. □

**Lemma 2** *If a set $S$ of $n$ points can be covered with $k$ lines ($k$ minimal or not), then: for any subset $R \subseteq S$ of at least $k+1$ collinear points (i.e., $|R| \geq k+1$ and $\forall \vec{r}_1, \vec{r}_2, \vec{r}_3 \in R : \vec{r}_1 \neq \vec{r}_2 \Rightarrow \exists \alpha \in \mathbb{R} : \vec{r}_3 = \alpha \cdot (\vec{r}_2 - \vec{r}_1) + \vec{r}_1$), the line through them is in the set of $k$ covering lines.*

**Proof.** Suppose the line through the points in $R$ was not among the $k$ lines covering $S$. Then the points in $R$ must be covered with at least $k+1$ lines, since no two points in $R$ can be covered with the same line. (The only line covering more than one point in $R$ is the one through all of them, which is ruled out.) Hence we need at least $k+1$ lines to cover the points in $R$. This contradicts the assumption that $S$ can be covered with $k$ lines. □

**Lemma 3** *If a set $S$ of $n$ points can be covered with $k$ lines ($k$ minimal or not), then: any subset of $S$ containing at least $k^2+1$ points must contain at least $k+1$ collinear points.*

**Proof.** Suppose there is a subset $R \subseteq S$ containing at least $k^2+1$ points, but not containing $k+1$ collinear points. Then each of the $k$ covering lines must contain at most $k$ points in $R$. Hence with these at most $k$ covering lines, each containing at most $k$ points, we can cover at most $k^2$ points. Thus we cannot cover $R$ (nor any superset of $R$, like $S$) with the $k$ lines. This contradicts the assumption that $S$ can be covered with $k$ lines. □

**Corollary 1** *If in any subset of $S$ containing at least $k^2+1$ points we do not find $k+1$ collinear points, we can conclude that $S$ cannot be covered with $k$ lines.*

**Lemma 4** *If a set $S$ of $n$ points can be covered with $l$ lines, but not with $l-1$ lines (i.e., if $l$ is the minimum number of lines needed to cover $S$) and $k \geq l$, then: if we generate all lines containing more than $k$ points, the total number of uncovered points will be at most $l \cdot k$.*

**Proof.** Let $R$ be the set of uncovered points in $S$ after all lines containing more than $k$ points have been generated. Since $S$ can be covered with $l$ lines and $R \subseteq S$, $R$ can be covered with $l$ (or fewer) lines. None of the lines covering points in $R$ can cover more than $k$ points in $R$ (as all such lines have already been generated). Hence there can be at most $l \cdot k$ points in $R$. □

## 3 General Procedure

Given a set $S$ of $n$ points in the plane, we already know (because of Lemma 1) that the minimum number $l$ of lines needed to cover $S$ is in $\{1, \ldots, \lceil \frac{n}{2} \rceil\}$. In our algorithm, we first check whether $l = 1$, which can obviously be decided in time linear in $n$. If the check fails (i.e., if the points in $S$ are not all collinear and thus $l \geq 2$), we try to increase the lower bound for $l$ by exploiting Lemmas 2 and 3, which (sometimes) provide us with means of proving that the set $S$ cannot be covered with a certain number $k$ of lines. In the first place, if for a given value of $k$ we find a subset $R \subseteq S$ containing $k^2 + 1$ points, but not containing $k+1$ collinear points, we can conclude that more than $k$ lines are needed to cover $S$ (because of Corollary 1). On the other hand, if we find $k + 1$ collinear points (details of how this is done are given below), we record the line through them (as it must be among the covering lines due to Lemma 2) and remove from $S$ all points covered by this line. This leads to a second possible argument: If by repeatedly identifying lines in this way (always choosing the next subset $R$ from the remaining points), we record $k$ lines while there are points left in $S$, we can also conclude that more than $k$ lines are needed to cover $S$.

We check different values of $k$ in increasing order (the exact scheme is discussed below), until we reach a value $k_1$, for which we fail to prove (with the means mentioned above) that $S$ cannot be covered with $k_1$ lines. On the other hand, we can demonstrate (in one of the two ways outlined above) that $S$ cannot be covered with $k_0$ lines, where $k_0$ is the largest value smaller than $k_1$ that was tested in the procedure. At this point we know that $l > k_0$.

Suppose that when processing $S$ with $k = k_1$, we identified $m_1$ lines, $m_1 \leq k_1$. We use a simple greedy algorithm to find $m_2$ lines covering the remaining points. (Note that $m_2$ may or may not be the minimum number of lines needed to cover the remaining points. Note also that $m_2 = 0$ if there are no points left to be covered.) As a consequence we know that $S$ can be covered with $m_1 + m_2$ lines (since we have found such lines) and thus that $k_0 < l \leq m_1 + m_2$. We show in [3] that $m_1 + m_2 \in O(l \log l)$ and thus that with the $m_1 + m_2$ lines we selected we obtained an $O(\log l)$ approximation of the optimum.

In a second step we may then go on and determine the exact value of $l$ by drawing on the found approximation (See the full version of paper in [3] for details).

Our proposed approximate and exact algorithms to solve the minimum line covering problem use the following two already known algorithms as subroutines:

1. An algorithm proposed by Guibas *et al.* [4], which finds all lines containing at least $k+1$ points in a set $S$ of $n$ points in time $O\left(\frac{n^2}{k+1} \log \frac{n}{k+1}\right)$.

2. An algorithm proposed by Langerman and Morin [7], which takes as input a set $S$ of $n$ points and an integer $k$, and outputs whether $S$ can be covered with $k$ lines in $O(nk + k^{2(k+1)})$ time.

### 3.1 Approximation for Minimum Line Covering

**Lemma 5** *We can approximate the minimum line covering problem within a factor of $O(\log l)$ in $O(n \log l)$ time if $l \leq \sqrt[4]{n}$.*

**Theorem 6** *We can approximate the minimum line covering problem within a factor of $O(\log l)$ in time $O(n \log l + l^4 \log l)$.*

**Corollary 2** *We can approximate the minimum line covering problem within a factor of $O(\log l)$ in $O(n \log l)$ time if $l \in O(\sqrt[4]{n})$.*

See the full paper [3] for the proofs of the above results.

### 3.2 Exact Minimum Line Covering

**Theorem 7** *The minimum line covering problem can be solved exactly in $O(n \log l + l^{2l+2})$ time. In particular, if $l \in O(\log^{1-\epsilon} n)$, the minimum line covering problem can be solved in $O(n \log l)$ time.*

**Theorem 8** *Given a set $S$ of $n$ points in the plane and an integer $k$, we can answer whether it is possible to find $k$ lines that cover all the points in the set in $O(n \log k + k^{2k+2})$ time.*

See the full paper [3] for the proofs for the above Theorems.

### 3.3 Producing the optimal set of lines

We also remark that after computing the optimal number of lines $l$, we can also produce the actual lines covering the input point set within the same time bounds. One way is to first use the algorithm proposed by Guibas *et al.* [4] to produce lines covering at least $l+1$ points. Let $l'$ denote the number of lines covering at least $l+1$ points and $n'$ the number of points left to be covered. We observe that at least one of the remaining $l-l'$ lines cover at least $\frac{n'}{l-l'}$ points. Let a line be called a *candidate* line if it covers at least $\frac{n'}{l-l'}$ points. Let us now compute the first candidate line: If $n' \leq 2(l-l')$ then any line covering at least two points can be included in the optimal solution. Otherwise, we tentatively (temporarily) remove the points covered by it and call Langerman and Morin's algorithm [7] to see whether the remaining points can be covered by $l-l'-1$ lines. Clearly, the candidate can be included in the optimal solution if and only if the answer is yes. Let $n_r$ denote the remaning points to be covered and $l_n$ denote the number of lines needed to cover $n_r$. We repeat the above algorithm and update $n_r$ and $l_n$ accordingly after each construction of a candidate line.

To calculate the time bound we show that there are at most $\frac{3}{2} \cdot (l-l')^2$ candidate lines. Any point can be covered by no more than $\frac{3}{2} \cdot (l-l')$ candidate lines. (The factor $\frac{3}{2}$ comes from the extreme case when $l-l' = \frac{n'}{3}$, so that each candidate line covers only three points and the point is covered by $\frac{n'-1}{2}$ candidates.) Hence, if we sum for each point the number of candidates it is covered by, we thus get an upper bound of $n' \cdot \frac{3}{2}(l-l')$. But we observe that this sum equals the sum we obtain by adding for each candidate line the number of points it covers. Since each candidate line covers at least $\frac{n'}{l-l'}$ points, the number of candidate lines cannot be larger than $(n' \cdot \frac{3}{2}(l-l'))/\frac{n'}{l-l'}$ and hence not larger than $\frac{3}{2}(l-l')^2$. Therefore we call Langerman and Morin's algorithm [7] at most $\frac{3}{2}(l-l')^2$ times before we produce one more optimal line. In subsequent calls to their algorithm, the number of optimal lines to be produced gets smaller and hence the time complexity gets smaller each time by at least a constant factor, since it is exponential in the number of optimal lines. This results in a geometric progression of the time complexity. Therefore the worst-case bound for the first call asymptotically dominates all subsequent calls.

### 3.4 Improving the time bound when $l \in \Omega(\log n)$

**Theorem 9** *For any input set of $n$ points, it can be decided whether there is a set of lines of cardinality at most $k$ covering the $n$ points in time $O(n \log k + (\frac{k}{2.2194\ldots})^{2k})$. Moreover, an optimal set of covering lines with minimum cardinality $l$ can be produced in time $O(n \log l + (\frac{l}{2.2194\ldots})^{2l})$*

**Proof.** See the full paper [3] of the proof. □

## 4 Lower Bound

In this section we give a lower bound on the time complexity for solving the minimum line cover problem. We make the assumption that the minimum number of lines $l$ needed to cover a set $S$ of $n$ points is at most $O(\sqrt{n})$. (For larger values of $l$ our lower bound may not be very interesting, since the best known upper bounds on the time complexity of the minimum line cover problem are exponential anyway.)

The main result we prove in this section is as follows:

**Theorem 10** *The time complexity of the minimum line cover problem is $\Omega(n \log l)$ in the algebraic decision tree model of computation.*

We prove Theorem 10 with a reduction from a special variant of the general set inclusion problem [1], which we call the $k$-Array Inclusion problem. Set inclusion is the problem of checking whether a set of $m$ items is a subset of the second set of $n$ items with $n \geq m$. Ben-Or [1] showed a lower bound of $\Omega(n \log n)$ for this problem using the following important statement.

**Statement 1** *If YES instances of some problem $\Pi$ have $N$ distinct connected components in $\Re^n$, then the depth of the real computation tree for this problem is $\Omega(\log N - n)$.*

Applying Statement 1 to the complement of $\Pi$, we get the same statement for NO instances. We define the $k$-array inclusion problem as follows:

**Definition 1** **$k$-Array Inclusion Problem:** *Given two arrays $A[1 \ldots k]$ of distinct real numbers and $B[1 \ldots m]$ of (not necessarily distinct) real numbers, $k \leq m$, $m+k=n$, determine whether or not each element in $B[1 \ldots m]$ belongs to $A[1 \ldots k]$.*

**Corollary 3** *Any algebraic computation tree solving $k$-array inclusion problem must have a depth of $\Omega(n \log k)$.*

**Proof.** This lower bound can be shown in a corresponding way as the lower bound for the set inclusion problem [1]. As already pointed out by Ben-Or, any computational tree will correctly decide the case when $A[1 \ldots k] = (1 \ldots k)$. The number of disjoint connected components, $N$, for YES instances of the $k$-array inclusion problem is $k^m$. This is because, in order to create a YES instance, for each element $m_i$ in $B[1 \ldots m]$, there are $k$ choices concerning which of the $k$ fixed elements in $A[1\ldots k]$, $m_i$ could be equal to. Since these choices are independent for each $m_i$, the total number of YES-instances becomes $k^m$. Applying Statement 1, we get a lower bound of $\Omega(m \log k)$, which is also $\Omega(n \log k)$, since $m > \frac{n}{2}$. □

To establish the lower bound in Theorem 10 for the minimum line cover problem, we convert (in linear time) the input of the $k$-array inclusion problem into a suitable input to the minimum line cover problem as follows: Each real number $a_i$, $1 \leq i \leq k$, in the array $A[1\ldots k]$ becomes $k$ points with coordinates $(a_i, j)$ ( in total we obtain $k^2$ points), $1 \leq j \leq k$ and each real number $b_j$ in the array $B[1\ldots m]$, $1 \leq j \leq m$, becomes a point with coordinates $(b_j, -j)$, all points are in two dimensional space. None of the constructed sets of $n = k + m$ points coincide. If we use any algorithm for the minimum line cover problem to solve the constructed instance, the output will be a set of lines covering these points. To obtain an answer to the $k$-array inclusion problem, we check whether the total number of lines, denoted by $l$, obtained for the minimum line cover problem is greater than $k$. If $l = k$ then each element in $B[1\ldots m]$ belongs to $A[1\ldots k]$, otherwise at least one element in $B[1\ldots m]$ does not belong to $A[1\ldots k]$. Since the $k$-array inclusion problem requires $\Omega(n \log k)$ time, it follows that the minimum line cover problem requires $\Omega(n \log k)$ time as well.

According to this construction, if it would be possible to compute the number $l$ in $o(n \log l)$ time, for some $l = O(\sqrt{n})$, then it would also be possible to solve the $k$-array inclusion problem in time $o(n \log k)$ for the case when $k = l$, which would contradict our lower bound for the $k$-array inclusion problem.

**Acknowledgment:** The authors wish to thank Dr. Christian Borgelt for his detailed reading and comments on the paper.

## References

[1] M. Ben-Or. Lower Bounds for Algebraic Computation Trees. *Proc. 15th Annual ACM Symp. on Theory of Computing*, 80–86. ACM Press, New York, NY, USA 1983

[2] H. Edelsbrunner, L. Guibas and J.Stolfi. Optimal Point Location in a Monotone Subdivision. *SIAM J. Comput.* 15:317–340. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA 1986

[3] M. Grantson and C. Levcopoulos. Covering a Set of Points with a Minimum Number of Lines. `http://www.cs.lth.se/~magdalene/lines.pdf`

[4] L. Guibas, M. Overmars, J. Robert. The Exact Fitting Problem in Higher Dimensions. *Computational Geometry: Theory and Applications*, 6:215–230. 1996

[5] D. Johnson. Approximation Algorithms for Combinatorial Problems. *J. of Comp. Syst. Sci.* 9:256-278. 1974

[6] V. Kumar, S. Arya, and H. Ramesh. Hardness of Set Cover With Intersection 1. *Proc. 27th Int. Coll. Automata, Languages and Programming,* LNCS 1853:624–635. Springer-Verlag, Heidelberg, Germany 2000

[7] S. Langerman and P. Morin. Covering Things with Things. *Proc. 10th Annual Europ. Symp. on Algorithms (Rome, Italy)*, LNCS 2461:662–673. Springer-Verlag, Heidelberg, Germany, 2002

[8] N. Megiddo and A. Tamir. On the Complexity of Locating Linear Facilities in the Plane. *Operation Research Letters* 1:194–197. 1982

[9] N. Sarnak, and R.E. Tarjan. Planar Point Location Using Persistent Search Tree. *Comm. ACM* 29:669-679. ACM Press, New York, NY, USA 1986

# Min-max-min Geometric Facility Location Problems

Jean Cardinal* Stefan Langerman†

## Abstract

We propose algorithms for a special type of geometric facility location problem in which customers may choose not to use the facility. We minimize the maximum cost incurred to a customer, where the cost itself is a minimum between two costs, according to whether the facility is used or not. We therefore call this type of location problem a *min-max-min* geometric facility location problem. As a first example, we describe the Closer Post Office problem, a generalization of the minimum spanning circle problem. We show that this problem can be solved in $O(n)$ randomized expected time. We also show that the proposed algorithm solves two other min-max-min geometric facility location problems. One, which we call the Moving Walkway problem, seems to be the first instance of a facility location problem using time metrics.

## 1 Introduction

In this work we study facility location problems in which customers make some decision on whether they have some interest in using the facility or not. The facility is defined as a geometric object, and customers are not interested in using it if it is too far from their own location.

We assume there are $n$ customers. If we denote by $x$ the facility to be located, then $C_x(i)$ is the cost incurred to the $i$th customer if she uses the facility, and $C_{\bar{x}}(i)$ is the cost if she does not use the facility. A *min-max-min* facility location problem is a problem of the form:

$$\min_x \max_{1\leq i\leq n} \min\{C_x(i), C_{\bar{x}}(i)\}.$$

An interesting application of this model is transportation facility location. A transportation facility might be for instance a bus line, a subway station, or an air connection between two airports. When setting up a new facility of this type, a company must take into account its usefulness, since customers that already have access to a closer or faster existing transportation facility will certainly not use the new one.

*Université Libre de Bruxelles, Computer Science Department, jcardin@ulb.ac.be

†Université Libre de Bruxelles, Computer Science Department, Chercheur qualifié FNRS. slanger@ulb.ac.be

### 1.1 Related Works

The simplest form of facility location is maybe the Weber problem (see for instance [7], Chapter 1), in which a point minimizing the sum of distances to $n$ other points is to be found. This problem dates back to the seventeenth century and many variations of it are still the subject of intense research nowadays.

Recently, Cabello et al. proposed algorithms for *reverse facility location* problems [3]. This term refers to *reverse nearest neighbor queries* in sets of points. Those queries receive a point as input and return the points from the set that have the query point as nearest neighbor. In reverse facility location, a new facility is located in such a way that the corresponding reverse nearest neighbor query returns the maximum number of points. The authors prove that this problem is 3SUM-hard and also propose algorithms for locating the facility with respect to additional min-max or max-min distance criteria. The problem we consider in Section 2 is a variant of this, which can be solved more efficiently. Reverse facility location can be considered as a discrete version of the problem studied in [5] of inserting, in a given set of points, a new point whose Voronoi cell has maximum size.

Several geometric optimization problems can be cast as finding minima on the upper envelope of a set of so-called *Voronoi surfaces*, generated by distance functions. In [8], it is proposed to construct completely this upper envelope to solve a family of problems, one of which is the minimum Haussdorff distance under translation between two point sets. General bounds on the description complexity of surface envelopes are given in Sharir and Agarwal's book on Davenport-Schinzel sequences [10]. They also propose algorithms for constructing the envelopes, which are used in the reverse facility location algorithms of [3]. The problems we consider in the following sections are all sufficiently simple so that we can avoid constructing the whole envelope of Voronoi surfaces.

In the min-max-min facility location problems we consider, the cost for a customer is the result of a (simple) minimization problem. If the facility is a transportation facility, then this minimum can be interpreted as a *time metric*. Time metrics have recently shown to be useful in the geometric analysis of transportation networks [1, 2]. The problem we consider in Section 4 is, to the authors' knowledge, the first instance of a facility location problem using time metrics.

### 1.2 Our Contributions

In Section 2, we define the Closer Post Office problem, a first, simple application of the proposed model. Suppose a new post office is to be installed in a city and we wish to minimize the maximum distance between any customer and the closest post office. If, for some customer, the new post office is not closer than the one she is used to go to, she will not use it and her cost will not vary. This is reminiscent of reverse facility location, except that the cost of all customers are taken into account, and not only the costs of the customers that actually use the facility. We observe that this problem boils down to finding a minimum height point on the upper envelope of a set of surfaces, each of which is the lower envelope of a horizontal plane and a cone.

In Section 3, we show that this problem is solvable in $O(n)$ randomized expected time.

In Section 4, we consider a transportation facility location problem, the Moving Walkway problem. It consists of finding the best location of a simple transportation facility, that is a moving walkway, modeled as an interval on the real line. Using the previous developments, we show that we can solve it in $O(n)$ randomized expected time as well.

In Section 5, we define a line location problem, the Highway problem. This problem can be interpreted as that of finding the optimal location of a highway, that customer might use to go quicker from point $s_i$ to point $t_i$. Its interpretation in the geometric dual plane (mapping lines to points and points to lines) leads to a simple formulation similar to the previous ones.

Finally, Section 6 presents higher-dimensional generalizations of the previous problems.

## 2 The Closer Post Office problem

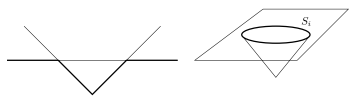
This problem is a generalization of the min-max center problem (or minimum spanning circle, see e.g. [9]) in which the distance function is the minimum between the actual distance to the facility and some constant.

**Definition 1** ($L_p$-Closer Post Office) *Given $n$ pairs $(s_i, t_i)$ of points in the plane, find a point $x^*$ which solves the following problem:*

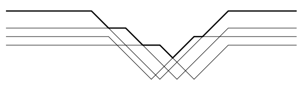
$$\min_{x \in \mathbb{R}^2} \max_{1 \le i \le n} \min\{d(s_i, t_i), d(s_i, x)\},$$

*where $d(.,.)$ is the $L_p$ distance function.*

We denote by $d_i(x) = \min\{d(s_i, t_i), d(s_i, x)\}$ the cost for customer $i$. The function $d(s_i, x)$ in the plane defines a cone centered on $s_i$. The function $d_i(x)$



(a) A single surface $f_i$.



(b) The upper envelope of the surfaces $f_i$.

Figure 1: Illustration of the objective function yielded by the $L_p$-Closer Post Office problem.

therefore defines a surface $f_i$ that is the lower envelope of the cone and the plane of equation $f(x) = d(s_i, t_i)$. We call $S_i$ the region of equation $d(s_i, t_i) \ge d(s_i, x)$, which is a scaled and translated version of the unit disk for the $L_p$ distance function that is used.

Since we minimize the maximum distance $d_i(x)$, this amounts to finding the minimum height point on the upper envelope of the set of surfaces $f_i$. This is depicted schematically on Figure 1.

## 3 A General Algorithm

In order to solve the Closer Post Office problem, we first reformulate it in terms of the surfaces $f_i$. We let $h_i = d(s_i, t_i)$ be the height of the surface $f_i$. We define the disks $S_i(h)$ for each surface $f_i$ and real number $h$ as the regions of equation $d(s_i, x) \le h$ if $h < h_i$ and $\mathbb{R}^2$ otherwise.

**Lemma 1** *Solving the* Closer Post Office *problem is equivalent to finding the minimum value $h^*$ of $h$ for which the intersection $\bigcap_{i=1}^{n} S_i(h)$ is nonempty.*

We first solve the decision problem consisting of verifying whether $h > h^*$. This amounts to checking whether a set of disks has a nonempty intersection, which can be done in $O(n)$ randomized expected time using Seidel's algorithm (see e.g. [6]).

In order to solve the corresponding optimization problem within the same time bounds, we use Timothy Chan's reduction [4] of an optimization problem to its decision version. For that purpose, we have to decompose the original problem in $m$ subproblems $\{Q_1, Q_2, \ldots, Q_m\}$ of size $n/c$ for some constants $m$ and $c$, such that the solution $h^*$ of the original problem is the maximum of the solutions of the $m$ subproblems.

We first make reasonable assumptions on the problem, namely that the surfaces $f_i$ are symbolically tilted and the points $s_i$ and $t_i$ are in some general position so that the minimum $h^*$ is on the intersection of a constant, say $k$, number of surfaces. To find a suitable subproblem decomposition, we partition the set of surfaces $\{f_i\}$ in $k+1$ disjoint subsets $F_1, F_2, \ldots, F_{k+1}$. Each subproblem $Q_j$ is defined the same way as our original problem, but only on the union of $k$ subsets $F_\ell$ of surfaces $f_i$. There are as many subproblems as there are $k$-subsets of the set $\{F_1, F_2, \ldots, F_{k+1}\}$, thus $m = k+1$. Each subproblem has size $\frac{k}{k+1}n$, thus we identify $c = (k+1)/k$. The minimum we look for is defined by $k$ surfaces, and this $k$-tuple must appear in at least one of the subproblems $Q_j$. In all the other subproblems, the minimum cannot be smaller, hence the solution is given by the maximum of the solutions of the $m$ subproblems. This gives us all necessary conditions for the reduction to work, and we can therefore solve the Closer Post Office problem using Seidel's algorithm.

**Theorem 2** *The $L_p$-Closer Post Office problem can be be solved in $O(n)$ randomized expected time.*

Note that this algorithm can be applied to any similar problem, in which we look for the minimum on the upper envelope of a set of surfaces, each being a plane with a convex "hole" of constant description complexity.

## 4 The Moving Walkway problem

The motivation for the next problem is the following. Suppose a new moving walkway is to be installed in a long corridor (for instance in the concourse of an airport). This walkway is to be used by people to go from one point of the corridor to another, and we wish to locate it so that it is the most useful. We model the corridor as the real line, source and destination points by pairs $(s_i, t_i)$ of real numbers with $t_i \geq s_i$, and the moving walkway by an interval $[a, b]$ on the line, with $b \geq a$. We denote by $v^{-1} > 1$ the speed on the moving walkway, and assume that the speed outside the moving walkway is 1. Customers can only enter the walkway at point $a$ and step down of it at point $b$. Our objective function is the maximum time needed to go from $s_i$ to $t_i$.

The time to go from any point $s$ of the line to any other point $t$ is really a time metric [1, 2], defined as the minimum between the difference $t - s$ and the following sum:

$$|s-a| + |t-b| + v(b-a).$$

The first two terms are the time needed to go from $s$ to the entrance of the walkway, and from the end of the walkway to $t$, respectively, while the third term is the time spent on the walkway. We can now define the problem.

**Definition 2** (Moving Walkway) *Given $n$ pairs $(s_i, t_i)$ of real numbers with $t_i > s_i$, and a real number $v \leq 1$, find a pair $(a^*, b^*)$ which solves the following problem:*

$$\min_{(a,b)\in\mathbb{R}^2 : b\geq a} \ \max_{1\leq i\leq n} d_i(a,b)$$

*where $d_i(a,b) = \min\{t_i - s_i, |s_i - a| + |t_i - b| + v(b-a)\}$.*

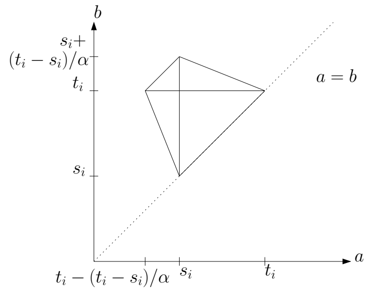


Figure 2: Edges of the surface defined by the function $d_i(a,b)$ in the Moving Walkway problem.

The following lemma characterizes the situations in which customers decide not to use the walkway. We denote by $\ell(I)$ the length of an interval $I$.

**Lemma 3** *The time metric $d_i(a,b)$ between $s_i$ and $t_i$ can be written as:*

$$d_i(a,b) = \begin{cases} t_i - s_i & \text{if } \ell([a,b]\cap[s_i,t_i]) \\ & \leq \alpha(b-a) \\ |s_i - a| + |t_i - b| & \\ +v(b-a) & \text{otherwise,} \end{cases}$$

*where $\alpha = (v+1)/2$.*

We now consider the function $d_i(a,b)$ in the plane $(a,b)$. This function defines a surface $f_i$ that has a structure similar to those appearing in the Closer Post Office problem. The surface $f_i$ is the lower envelope of a horizontal plane corresponding to the inequality $d_i(a,b) \leq t_i - s_i$, and a piecewise linear surface made of four patches. The vertices of $f_i$ are the following: $d_i(s_i, t_i) = v(t_i - s_i)$, $d_i(t_i, t_i) = d_i(s_i, s_i) = t_i - s_i$, and $d_i(s_i, s_i + (t_i - s_i)/\alpha) = d_i(t_i - (t_i - s_i)/\alpha, t_i) = t_i - s_i$. The projection of the edges of the surface $f_i$ on the plane $(a,b)$ is depicted on Figure 2. If the speed on the walkway is infinite, that is when $v = 0$, then the surface $f_i$ is the lower envelope of a horizontal plane and a rectilinear cone, as those appearing in the $L_1$-Closer Post Office problem.

**Lemma 4** *The Moving Walkway problem with $v = 0$ is a special case of the $L_1$-Closer Post Office problem.*

It can be checked that the algorithm described in the previous section applies to this problem, even when $v > 0$.

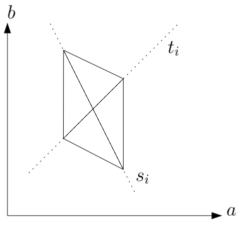


Figure 3: Edges of the surface defined by the function $d_i(a,b) = \min\{d(s_i,t_i), d_v(s_i,L) + d_v(L,t_i)\}$ in the HIGHWAY problem, where the line $L$ has equation $y = ax + b$. The function has value 0 at the intersection of the lines defined by the points $s_i$ and $t_i$ in the dual plane, and $d(s_i,t_i)$ outside the parallelogram.

**Theorem 5** *The* MOVING WALKWAY *problem can be be solved in $O(n)$ randomized expected time.*

## 5 The HIGHWAY problem

Our last problem is that of locating a line $L : y = ax+b$ in the plane. We denote by $d_v(x,L)$ the vertical distance between the point $x$ and the line $L$.

**Definition 3** (HIGHWAY) *Given $n$ pairs $(s_i,t_i)$ of points in the plane, find a line $L^*$ which solves the following problem:*

$$\min_L \max_{1\le i\le n} \min\{d(s_i,t_i), d_v(s_i,L) + d_v(L,t_i)\}$$

This problem can be interpreted as that of locating a highway, which customers may use to go from point $s_i$ to point $t_i$. When analyzed in the dual plane mapping lines to points and points to line, this problem can be shown to be very similar to the previous two ones, for it also consists of finding a minimum on the upper envelope of a set of surfaces, each of which is defined as the lower envelope of a plane and a cone, as depicted on Figure 3. The previous algorithm applies again in this situation.

**Theorem 6** *The* HIGHWAY *problem can be be solved in $O(n)$ randomized expected time.*

## 6 Higher-dimensional variants

Clearly, the algorithm designed to solve the CLOSER POST OFFICE problem can also be used to solve the analogous problem in $k > 2$ dimensions instead of 2. The decision problem then consists of checking whether an intersection of $k$-dimensional balls is empty, which can be done in linear time as long as $k$ remains constant. The 2-dimensional extension of the MOVING WALKWAY problem is of particular interest. It could model a situation in which for instance a new public transport connection is to be set up between two points. The problem is the following.

**Definition 4** ($2D$-$L_p$-MOVING WALKWAY) *Given $n$ pairs $(s_i,t_i)$ of points in the plane, and a real number $v \le 1$, find a pair $(a^*,b^*)$ which solves the following problem:*

$$\min_{(a,b)\in\mathbb{R}^2\times\mathbb{R}^2} \max_{1\le i\le n} d_i(a,b)$$

*where $d_i(a,b) = \min\{d(t_i,s_i), d(s_i,a) + d(t_i,b) + vd(a,b)\}$ and $d(.,.)$ is the $L_p$ distance function.*

In order for the algorithm to work, we have to make sure that the decision problem can be solved in linear time. We define the regions $S_i(h)$ as $\mathbb{R}^4$ if $d(s_i,t_i) \le h$ and as

$$\{(a,b) \in \mathbb{R}^4 : d(s_i,a) + d(b,t_i) + vd(a,b) \le h\}$$

otherwise. The decision problem amounts to checking whether the intersection of the regions $S_i(h)$ is empty. Those regions can be shown to be convex, hence the decision problem is a convex programming problem in 4 dimensions. This can be solved using a randomized algorithm in $O(n)$ time. From the previous developments, the optimization problem can be solved within the same time bound.

## References

[1] M. Abellanas, F. Hurtado, and B. Palop. Transportation networks and Voronoi diagrams. In *Proc. International Symposium on Voronoi Diagrams in Science and Engineering*, 2004.

[2] O. Aichholzer, F. Aurenhammer, and B. Palop. Quickest paths, straight skeletons, and the city Voronoi diagram. *Discrete & Computational Geometry*, 31(1):17–35, 2004.

[3] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura. Reverse facility location problems. In *Proc. 17th Canadian Conference on Computational Geometry (CCCG'05)*, pages 68–71, 2005.

[4] T. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22:547–567, 1999. (SoCG special issue).

[5] O. Cheong, A. Efrat, and S. Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 1098–1107, 2004.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer-Verlag, 1997.

[7] Z. Drezner and H. W. Hamacher, editors. *Facility Location: Applications and Theory*. Springer-Verlag, 2001.

[8] K. Kedem, D. P. Huttenlocher, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete Comput. Geom.*, 9(3):267–291, 1993.

[9] J.-M. Robert and G. T. Toussaint. Computational geometry and facility location. In *Proc. International Conf. on Operations Research and Management Science*, volume B, pages 1–19, Dec. 1990.

[10] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

# Proximity structures in the fixed orientation metrics

Christian Wulff-Nilsen*

## Abstract

We present algorithms computing two types of proximity structures in the plane with a fixed orientation metric. Proximity structures have proven useful for Steiner tree heuristics in the Euclidean plane and may play a similar role for the fixed orientation metrics where Steiner trees are important in the area of VLSI design. We show how to find an all nearest neighbour graph $NNG(Z)$ of a set $Z$ of $n$ points in $O(an \log n)$ time using $O(n)$ space where $a$ is the number of fixed orientations. The algorithm does not use the Voronoi diagram of $Z$. We present an algorithm that computes the Gabriel graph $GG(Z)$ of $Z$ in $O(an \log n)$ time using $O(an)$ space under the assumption that no three points of $Z$ are on a line parallel to one of the boundary edges of a $\sigma$-circle where $\sigma$ is the set of fixed orientations. We show that if this assumption is not satisfied then $GG(Z)$ may contain $\Omega(n^2)$ edges. Both algorithms are optimal for constant $a$.

## 1 Introduction

Let $\sigma$ be a set of $a$ angles $\alpha_1, \ldots, \alpha_a$ sorted counter-clockwise. These angles are called *(fixed) $\sigma$-orientations* and a line, halfline, or line segment with a $\sigma$-orientation is said to be *$\sigma$-oriented.* The *$\sigma$-distance* $d_\sigma$ between two points is the length of a shortest path of $\sigma$-oriented line segments between the points and $d_\sigma$ is called the *$\sigma$-metric.* We refer to the $\sigma$-metric as a *fixed orientation metric.* We will sometimes write $|pq|_\sigma$ instead of $d_\sigma(p, q)$.

In recent years, fixed orientation metrics have played an important role in the area of VLSI design. This is due to the fact that the orientations of wires on a print are typically restricted to a finite set of fixed orientations. In VLSI routing, an important objective is to minimize the total length of an interconnection. This involves computing a Steiner tree. Since the Steiner tree problem is NP-hard, heuristics can be applied. Proximity structures have proven efficient for Steiner tree heuristics in the Euclidean plane [3] and could possibly play a similar role for Steiner tree heuristics in fixed orientation metrics.

We consider two types of proximity structures in the $\sigma$-metric. The first is an *all nearest neighbour graph $NNG(Z)$* of $Z$ where $Z$ is a finite set of $n$ points or *terminals* in the plane. This graph has an edge $(z_1, z_2)$ if and only if $z_2$ is a *nearest neighbour* of $z_1$, i.e. $d_\sigma(z_1, z_2) = \min_{z \in Z\setminus\{z_1\}} d_\sigma(z_1, z)$, or if $z_1$ is a nearest neighbour of $z_2$. If a terminal has more than one nearest neighbour, only one of them is picked as a nearest neighbour.

For $c \in \mathbb{R}^2$ and $r > 0$, the *$\sigma$-circle* $C_\sigma(c, r)$ with center $c$ and radius $r$ is the set of points having $\sigma$-distance at most $r$ to $c$, i.e. $C_\sigma(c, r) = \{p \in \mathbb{R}^2 | d_\sigma(c, p) \leq r\}$. The second proximity structure we consider is the *Gabriel graph $GG(Z)$ of $Z$.* This graph has an edge $(z_1, z_2)$ if and only if the $\sigma$-circle with center $c = \frac{1}{2}(z_1 + z_2)$ and radius $|cz_1|_\sigma = |cz_2|_\sigma$ contains no terminals in its interior.

We present algorithms computing $NNG(Z)$ and $GG(Z)$ in the $\sigma$-metric.

The organization of the paper is as follows. In section 2, we make various definitions and present some basic properties related to the $\sigma$-metric. In section 3, we show how to find $NNG(Z)$ in $O(an \log n)$ time using $O(n)$ space. We also show how to compute a similar all nearest neighbour graph and, using this, we present an $O(an \log n)$ time and $O(an)$ space algorithm computing $GG(Z)$ in section 4. To obtain these bounds, we make a simplifyng assumption about $Z$ since otherwise, $GG(Z)$ may contain $\Omega(n^2)$ edges. Our algorithms are optimal for constant $a$ as we show in section 6. Finally, we make some concluding remarks in section 7.

## 2 Definitions and basic properties

Associate with each $z \in Z$ the region of points not farther from $z$ than from any other terminal in the $\sigma$-metric. This region is a (possibly unbounded) polygon [2] and is referred to as the *Voronoi polygon of $z$.* Since Voronoi polygons are not always disjoint, they need not define a partition of the plane.

We define a *Voronoi diagram of $Z$, $Vor(Z)$,* to be a partition of the plane into regions such that each region contains some $z \in Z$ together with (some of the) points not farther from $z$ than from any other terminal in $Z$. The region containing $z$ is a (possibly unbounded) polygon [2] called the *Voronoi region (of $z$)* and the line segments defining the boundary of a Voronoi region are called *Voronoi edges.* A Voronoi diagram $Vor(Z)$ of $Z$ can be found in $O(an \log n)$ time using $O(an)$ space [2].

Given a point $p$, a *$\sigma$-cone of $p$* is the region bounded

*Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark, koolooz@diku.dk

by halflines emanating from $p$ having orientations $\alpha_i$ and $\alpha_{i+1}$ respectively for some $i$ (if $i = a$ then the orientations are $\alpha_a$ and $\alpha_1$ respectively). We order the $\sigma$-cones of $p$ counter-clockwise starting with the $\sigma$-cone having orientations $a_1$ resp. $a_2$.

Let $q$ be another point. The union of $\sigma$-oriented lines through $p$ resp. $q$ partitions the plane into regions called *fields*. The *bisector of p and q* is the set of points $r$ such that $d_\sigma(p,r) = d_\sigma(q,r)$. When the bisector of $p$ and $q$ is restricted to a field, it is either empty, a line segment, or the whole field [2]. In the latter case, we will refer to the field as a *bisector field of p and q*.

We will need another type of all nearest neighbour graph, denoted $NNG(M, M')$, where each point in $M$ is incident to a nearest neighbour among points in $M'$ for finite point sets $M$ and $M'$.

As we shall see later, $GG(Z)$ may contain $\Omega(n^2)$ edges unless we make some simplifying assumption about $Z$. We will show that one such assumption is the following: no three terminals are on the same line parallel to a boundary edge of a $\sigma$-circle. If this assumption is satisfied, we say that the terminals are in *general position*.

## 3 All nearest neighbour graph

In this section, we show how to find $NNG(Z)$ in $O(an \log n)$ time using $O(n)$ space. This is done without computing $Vor(Z)$, as opposed to the algorithm suggested in [2]. The idea is to linearly transform $Z$ by mapping $\sigma$-cones to north-east (NE) quadrants in such a way that finding a nearest neighbour in a $\sigma$-cone is equivalent to finding a nearest NE-neighbour in the transformed terminal set in the $L_1$-metric. Nearest NE-neighbours are then found using the algorithm suggested in [1].

We will briefly describe the algorithm in [1] since we need to modify it later. It finds nearest north-east neighbours in the $L_1$-metric by partioning the given point set into a left and a right half, recursively finding nearest NE neighbours in the two halves, and then finding nearest NE neighbours of points in the left half among points in the right half. To do the latter, the algorithm keeps track of three pointers, *left*, *right*, and *min*. During the course of the algorithm, *left* advances down the list of points in the left half, *right* advances down the list of candidate nearest NE neighbours of *left* in the right half, and *min* keeps track of the nearest NE neighbour of *left* in the right half found so far. Pointer *left* makes only one pass through the points in the left half. Similarly, *right* makes only one pass through the points in the right half.

Let $K$ be the $i$th $\sigma$-cone of a point $p = (p_x, p_y)$ and let $q = (q_x, q_y)$ be a point in $K$. By rotating if necessary, we may assume that the right leg of $K$ is aligned with the $x$-axis. Now, letting $s = \cot\theta$, where $\theta = \alpha_{i+1} - \alpha_i$, we have (see Figure 1)

$$d_\sigma(p,q) = (q_x - p_x) + (q_y - p_y)\left(\sqrt{s^2+1} - s\right).$$

Define $T_i : \mathbb{R}^2 \to \mathbb{R}^2$ by

$$T_i(x,y) = \left(\frac{x - sy}{\sqrt{s^2+1}}, y\right).$$

The following lemma shows that $T_i$ is the linear transformation we are looking for.

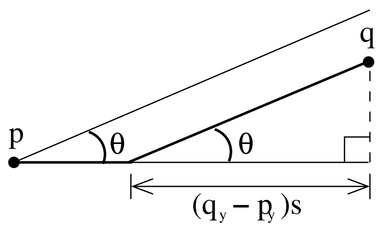


Figure 1: The $\sigma$-distance from $p$ to $q$ is the sum of lengths of the two $\sigma$-oriented line segments marked in bold.

**Lemma 1** *With the above definitions, $T_i$ is linear and maps $K$ to the first quadrant of $T_ip$. If $T_iq$ is in the first quadrant of $T_ip$ then $q \in K$. If $q_1, q_2 \in K$ then $d_\sigma(p,q_1) \le d_\sigma(p,q_2)$ if and only if $L_1(T_ip, T_iq_1) \le L_1(T_ip, T_iq_2)$.*

The algorithm computing $NNG(Z)$ is as follows. For $i = 1, \ldots, a$, we make a call to the algorithm in [1] on $T_i(Z)$. By Lemma 1, this gives us nearest neighbours in the $i$th $\sigma$-cones of terminals in $Z$. To ensure $O(n)$ space requirement, we maintain only the nearest neighbour of each terminal found so far during the iterations.

The time spent in each of the $a$ iterations is bounded by the time spent in the algorithm in [1] which is $O(n \log n)$. Thus, the total running time of our algorithm is $O(an \log n)$. The space used is $O(n)$, the amount required by the algorithm in [1].

In order to construct $NNG(M, M')$, where $|M| = m$ and $|M'| = m'$, we need to modify the update of pointers *left* and *right* in the algorithm in [1]. We do this as follows. The input point set is $M \cup M'$. Since we need to find nearest neighbours of points in $M$ among points in $M'$ pointer *left* needs to skip points in $M'$ and pointer *right* needs to skip points in $M$. With this modification, we can construct $NNG(M, M')$ in $O(a(m+m') \log(m+m'))$ time using $O(m+m')$ space.

## 4 Gabriel graph

Next, we consider the Gabriel graph $GG(Z)$ of $Z$. We will show how to find this graph in $O(an \log n)$ time using $O(an)$ space. We assume that a supergraph

$S(Z)$ of $GG(Z)$ containing $O(n)$ edges is given. In section 5, we present an algorithm that finds such a supergraph in $O(an \log n)$ time and $O(an)$ space. However, we assume that terminals are in general position since otherwise, $GG(Z)$ may contain $\Omega(n^2)$ edges, see Figure 2.

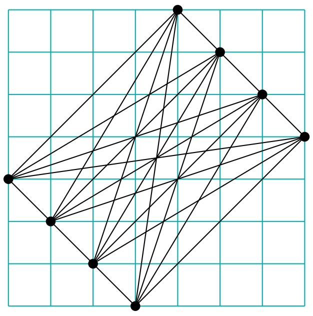


Figure 2: $GG(Z)$ with a quadratic number of edges, shown in the rectilinear metric.

Let $M$ be the set of midpoints of edges of the supergraph $S(Z)$. An edge $e = (z_1, z_2) \in S(Z)$ belongs to $GG(Z)$ if and only if a nearest neighbour terminal of midpoint $m$ of $e$ is no closer to $m$ than $z_1$ and $z_2$. From this it follows that, given $NNG(M, Z)$, which can be constructed in $O(an \log n)$ time using $O(n)$ space, determining whether $e$ belongs to $GG(Z)$ takes $O(1)$ time. Hence, edges of $S(Z)$ not belonging to $GG(Z)$ can be discarded in $O(n)$ time. This shows that $GG(Z)$ can be found in $O(an \log n)$ time using $O(an)$ space.

## 5 A supergraph of $GG(Z)$

In the Euclidean metric, the *Delaunay graph of Z* is the straight-line dual of the Voronoi diagram of $Z$. The Delaunay graph of $Z$ contains the Gabriel graph of $Z$. Unfortunately, this does not always hold in the $\sigma$-metric since a Voronoi region need not equal the corresponding Voronoi polygon, see Figure 3. The algorithm in [2] constructing $Vor(Z)$ does not always pick an entire bisector field but merely one of the halflines bounding this field when constructing the boundary of a Voronoi region $P$. Hence, the bisector parts bounding $P$ will not always be included in $P$ and the straight-line dual of $Vor(Z)$ will therefore not always contain $GG(Z)$.

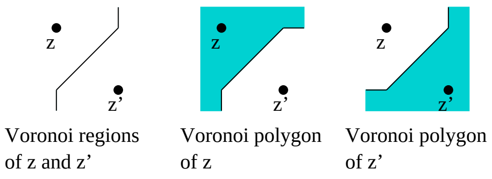


Figure 3: Voronoi regions and Voronoi polygons of terminals $z$ and $z'$, shown in the rectilinear metric.

To remedy this, we expand each Voronoi region of $Vor(Z)$ to its corresponding Voronoi polygon by including those parts of the bisector fields that belong to the Voronoi polygon. In the following, assume that $Vor(Z)$ is found using the algorithm in [2]. In the $\sigma$-metric, we define the *Delaunay graph $DG(Z)$ of $Z$* to be the graph having an edge between each pair of terminals whose Voronoi polygons overlap (possibly only on the boundaries). We will later see that $DG(Z)$ can be used as a supergraph $S(Z)$.

In order to efficiently construct $DG(Z)$, we will need a few results. Suppose Voronoi edges of each Voronoi region are directed clockwise. This involves replacing each edge of $Vor(Z)$ by two oppositely directed edges. Let $e = (p, q)$ be a Voronoi edge of the Voronoi region $P$ of a terminal $z_1$ and let $(q, p)$ belong to the Voronoi region $P'$ of a terminal $z_2$. The following lemma gives a necessary and sufficient condition for expanding $P$ at $e$.

**Lemma 2** *With the above definitions, $e$ bounds a bisector field $F$ of $z_1$ and $z_2$ such that $F$ does not intersect the interior of $P$ if and only if there are two lines $l$ and $l'$ with orientations $\alpha_i$ and $\alpha_{i+1}$ respectively for some $i$ such that $e, z_2, p \in l$ and $z_1, p \in l'$ (Figure 4).*

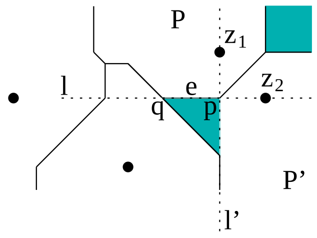


Figure 4: The situation in Lemma 2.

Next, we show that we only need to consider Voronoi regions adjacent to $P$ when expanding.

**Lemma 3** *Let $z_1$, $z_2$, and $z_3$ be distinct points. If $F_1$ is a bisector field of $z_1$ and $z_2$ and $F_2$ is a bisector field of $z_1$ and $z_3$ then $F_1 \cap F_2 = \emptyset$.*

We can quickly find those Voronoi edges of the neighbouring Voronoi region $P'$ that belong to the expanded $P$ as the next lemma shows.

**Lemma 4** *With the above definitions, if $e$ bounds a bisector field $F$ of $z_1$ and $z_2$ such that $F$ does not intersect the interior of $P$ then the Voronoi edges of $P'$ intersected by $F$ occur sequentially when walking around the Voronoi edges of $P'$ starting in $p$.*

**Theorem 5** *Given $Vor(Z)$, $DG(Z)$ can be constructed in $O(an)$ time using $O(an)$ space and $DG(Z)$*

*consists of $O(n)$ edges. Furthermore, $GG(Z) \subseteq DG(Z)$.*

**Proof.** We construct $DG(Z)$ from $Vor(Z)$ by initializing the edge set of $DG(Z)$ to the empty set and doing the following for each Voronoi region $P$ of $Vor(Z)$. Let $z_1$ be the terminal of $P$. For each edge $e = (p, q)$ of $P$, let $z_2$ be the terminal of the Voronoi region $P'$ sharing edge $e$ with $P$. We check if $e$ bounds a bisector field $F$ of $z_1$ and $z_2$ such that $F$ intersects the interior of $P$ (Lemma 2). If so, we add edge $(z_1, z_2)$ to $DG(Z)$. Otherwise, we need to expand $P$ at $e$. We do not do this explicitly since we only need to construct $DG(Z)$. Instead, we make a counter-clockwise detour into $P'$, starting in $q$. Each edge in this detour is adjacent to the Voronoi region of a terminal $z_3 \neq z_2$. We add $(z_1, z_3)$ to $DG(Z)$ if $(z_1, z_3)$ is not already in $DG(Z)$. The detour stops when we encounter an edge not intersecting $F$ or if there are no more edges in the tour (this may happen when $P'$ is unbounded).

The correctness of the above algorithm follows from Lemma 3 and Lemma 4. The running time is bounded from above by the number of edges we traverse. An (undirected) edge $e$ of $Vor(Z)$ is traversed at most four times, namely once in each of the two traversals of Voronoi regions adjacent to $e$ and, by Lemma 3, at most twice in detours. Thus, the running time is $O(an)$. The space requirement is clearly $O(an)$.

Let $Vor'(Z)$ be the graph obtained from $Vor(Z)$ by merging edges of $Vor(Z)$ meeting in degree two vertices into one edge. Since no vertex of $Vor'(Z)$ has degree less than three, the number of edges in this graph is $O(n)$ by Euler's formula (the unbounded faces are easily handled). In the above algorithm, an edge traversal of a Voronoi region or a detour in $Vor(Z)$ induces an edge traversal in $Vor'(Z)$. An argument similar to the one above shows that the number of times we traverse any edge of $Vor'(Z)$ is bounded by a constant. Thus, $DG(Z)$ contains $O(n)$ edges.

Let $e = (z_1, z_2)$ be an edge of $GG(Z)$ and let $m = (z_1 + z_2)/2$ be the midpoint of $e$. Since no terminals are strictly closer to $m$ than $z_1$ and $z_2$ it follows that the Voronoi polygons (the expanded Voronoi regions) of $z_1$ and $z_2$ overlap in $m$, hence $e \in DG(Z)$. □

## 6 Optimality of algorithms

Finally, we show that for fixed $a$, the two algorithms presented are optimal.

Space requirement is clearly optimal. That our $GG$-algorithm has optimal running time follows from reduction of sorting. Let $c_1, \ldots, c_n$ be $n$ distinct real numbers. Consider the corresponding set $Z$ of $n$ terminals $z_{c_1}, \ldots, z_{c_n}$, where $z_{c_i} = (c_i, 0)$ for $i = 1, \ldots, n$. Let $\tau$ be the permutation of $c_1, \ldots, c_n$ such that $\tau(c_1) < \tau(c_2) < \cdots < \tau(c_n)$ and let $G = (Z, E)$ where $E$ is the set of edges $(z_{\tau(c_i)}, z_{\tau(c_{i+1})})$ for $i = 1, \ldots, n-1$. Then $GG(Z) = G$, showing the optimality of the $GG$-algorithm.

Now, with $Z$ as above, construct $NNG(Z)$. This graph is a collection of connected components, each of which corresponds to a sublist of the sorted list of numbers. Letting $Z' \subseteq Z$ be the set of left endpoint of each sublist, the fact that each terminal has a nearest neighbour implies $|Z'| \leq \frac{1}{2}|Z|$, see figure 5. We repeat the procedure on $NNG(Z')$. Eventually we

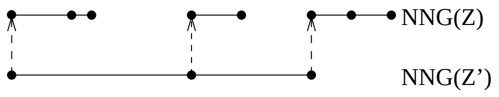


Figure 5: We can use $NNG(Z)$ to sort numbers.

end up with a single component. Using the generated graphs, we find the sorted list of $n$ numbers in linear time. If we could construct $NNG(Z)$ in $o(n \log n)$ time, we could construct the above sequence of graphs in $o(n \lg n)$ time, a contradiction.

## 7 Conclusion

We presented algorithms computing an all nearest neighbour graph $NNG(Z)$ and the Gabriel graph $GG(Z)$ for a set $Z$ of $n$ points in the plane with a fixed orientation metric. For a constant number of fixed orientations, both algorithms have $O(n \log n)$ optimal running time and $O(n)$ optimal space requirement. We assumed that no three points of $Z$ are on the same line parallel to a boundary edge of a $\sigma$-circle when constructing $GG(Z)$ and showed that $GG(Z)$ may contain $\Omega(n^2)$ edges in general.

It should be possible to extend both algorithms to the *weighted fixed orientation metrics* in which each fixed orientation has an associated weight.

Finally, it should be possible to relax the general position assumption without affecting the asymptotic time and space bounds of our algorithms by only requiring the following: at most a constant number of terminals are on the same line parallel to a boundary edge of a $\sigma$-circle.

## References

[1] Leo J. Guibas and Jorge Stolfi. On computing all north-east nearest neighbors in the $L_1$ metric. Information Processing Letters 17 (1983) 219-223, North-Holland.

[2] P. Widmayer, Y. F. Wu and C. K. Wong. On some distance problems in fixed orientations. Siam J. Comput. Vol. 16, No. 4, August 1987.

[3] M. Zachariasen and P. Winter. Concatenation-Based Greedy Heuristics for the Euclidean Steiner Tree Problem. Algorithmica (1999) 25: 418-437.

# Randolph's Robot Game is NP-complete!

Birgit Engels[*] Tom Kamphans[†]

## Abstract

We introduce a new type of movement constraints for a swarm of robots in a grid environment. This type is inspired by Alex Randolphs board game *Ricochet Robot* and may be used to model robots with very limited abilities for self localization: We assume that once a robot starts to drive in a certain direction, it does not stop its movement until it hits an obstacle wall or another robot. We show that the question, whether a given cell can be reached is NP-complete for arbitrary environments.

A Java applet for simulating robot swarms moving with these constraints can be found in
`http://www.geometrylab.de/RacingRobots/`

**Keywords:** Robot navigation, cellular environments, navigation errors, robot swarms, NP-completeness.

## 1 Introduction

Robot motion planning has received a lot of attention both in computational geometry and in robotics; see, for example, the surveys [3, 16, 11], or the books [15, 19, 6].

In this paper, we consider a quite simple model for the robots and their environment: The robots are short sighted and the surrounding is subdivided by a rectangular integer grid; that is, the robots move in a cellular environment, similar to a chessboard or squared writing paper.

Environments with a grid structure were considered in different settings. Icking et al. [12] studied the exploration problem (also known as covering) of a simple grid polygon (i.e., a polygon with no obstacles inside). They gave a lower bound of $\frac{7}{6}$ and a $\frac{4}{3}$-competitive exploration strategy for this problem. The case of a polygon with obstacles was considered by Icking et al. [10]—see also [14]—and independently by Gabriely and Rimon [9]. Itai et al. [13] showed that the corresponding offline problem is NP-hard. Betke et al. [4] and Albers et al. [1] studied the piecemeal exploration problem, where the robot has to return to the start cell every now and then. Cellular environments were also considered from a more practical point of view; see, for example, Moravec and Elfes [17].

Swarms of robots have been studied intensely. See, for example, Bruckstein et al. [5]. Arkin et al. [2] considered the *freeze-tag problem* (i.e., the question how to 'wake up' an initially inactive swarm of robots).

Figure 1: The board game *Ricochet Robots* by Alex Randolph.

Another interesting model for robots moving around in cellular environments was inspired by the board game *Ricochet Robots* by Alex Randolph [18], see Figure 1. The interesting part of the game are the rules to move a robot: A robot can move in one of the four directions (north, east, south, or west), but once it has chosen a direction it continues to move in this direction until it hits an obstacle or another robot. Thus, it is often necessary to move robots that serve as guides to stop the movement of another robot on an appropriate cell. See, for example, Figure 2: The task is to move the robot $\otimes$ to the cell marked with $\diamond$. To permit this movement, the robot $\oplus$ has to move to $a$, so three moves are necessary to solve the task.

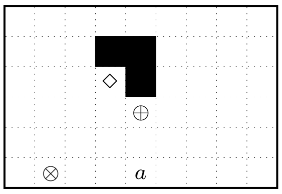


Figure 2: Example: the robot $\oplus$ has to move to $a$ to allow the robot $\otimes$ a movement to $\diamond$.

This model can be used for a swarm of robots. Each of them has a very restricted orientation: Even if the robots have a map of their environment, a robot that touches a wall knows only, which wall in the environment it touches, but as soon as the robot leaves the

[*]Universität zu Köln, Zentrum für angewandte Informatik (ZAIK), 50831 Köln, Germany. engels@zpr.uni-koeln.de

[†]University of Bonn, Institute of Computer Science I, 53117 Bonn, Germany. kamphans@cs.uni-bonn.de

wall it has no chance to locate itself. Therefore, it continues its movement until it hits another wall or another robot. However, the robots are able to communicate with each other, or all of them are controlled by the same computer. Apart from the best strategy to solve Randolph's game, an interesting question is, whether there is an upper bound for the number of robots, such that every cell can be reached by at least one robot. In this paper we show that the reachability problem in arbitrary environments is NP-complete.

## 2 Preliminaries

We assume that the robot's environment is subdivided by a rectangular integer grid. We call a reachable basic block in the environment a *cell*, and the set of all cells that can be reached by the robots a *grid polygon*, or polygon for short. An unreachable block is called an *obstacle* or *hole*.

We assume that the environment is populated by a system $R$ of $N$ robots $r_1, \dots, r_N$. The robots start either from a common start cell $s$ or from a given start configuration $S = (s_1, \dots, s_N)$ inside the polygon. The sensors of a robot provide the information, which of the four neighbors of the currently occupied cell belong to the polygon and which ones do not. Furthermore, the robot is able to recognize neighboring cells occupied by another robot. A robot can enter an unoccupied polygon cell, but once it started to drive in a certain direction, it will continue the movement until it hits a wall (i.e., an obstacle) or another robot. We assume that in any point of time at most one robot moves.

In spite of the very basic sensors, the robots are either able to communicate with each other or they are steered by a common controlling unit. However, the robot system $R$ provides enough memory to store a map of the environment and some additional information.

## 3 Reachability in Arbitrary Polygons

Let the *Reachability Problem* be defined as follows: Given an arbitrary grid polygon $P$, a system of $N$ *Randolph robots* $r_1, \dots, r_N$, a start configuration $S = (s_1, \dots, s_N)$, and a target cell $t$. Is one of the robots able to reach $t$—probably with the help of the other robots?

In this section, we show that the Reachability Problem is NP-complete by reducing the well known satisfiability problem with three literals per clause (3SAT) to the Reachability Problem. Let us recall the 3SAT Problem: Given a boolean expression, $\alpha$, consisting of $m$ clauses $C_1, \dots, C_m$ over $n$ variables $X_1, \dots, X_n$, where each clause consists of three literals; that is, $\alpha = C_1 \wedge \dots \wedge C_m$ with $C_i = (L_{i1} \vee L_{i2} \vee L_{i3})$ and $L_{ij} \in \{ X_\ell, \neg X_\ell;\ 1 \le \ell \le n \}$. Is there a truth assignment for $X_1, \dots, X_n$ such that $\alpha$ is fulfilled?

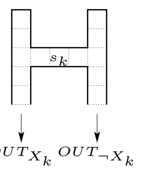


Figure 3: A fork polygon.

Given an expression $\alpha$, we construct a polygon, $P(\alpha)$. In the following, we give a brief description of the construction.[1] The robot system $R$ consists of $n+1$ robots, $r_0, r_1, \dots, r_n$, where $n$ is the number of variables in the 3SAT instance. The robot $r_0$ plays a special role: it starts on a special start cell $s_0$ and its purpose is to reach $t$. Note, that $r_0$ is the only robot that may reach $t$, if $t$ is reachable at all. Each of the other robots $r_k$, $1 \le k \le n$, represents a variable $X_k$ and its truth assignment. Thus, we call these robots *literal robots*. A literal robot $r_k$ starts at a cell $s_k$ in a special fork-shaped (sub-)polygon called *fork polygon* $fp_k$, see Figure 3. As soon as $r_k$ leaves the fork polygon, it is impossible for $r_k$ to return to $s_k$ and to enter the other vertical passage. This property ensures the consistency of the truth assignment for the variables in $\alpha$. By convention, we assign $X_k := 1$ if the robot enters the left-hand vertical passage of the corresponding fork polygon, and $X_k := 0$ otherwise. We denote the former passage with $X_k$*-passage* and the latter with $\neg X_k$*-passage* of $fp_k$. We have:

**Lemma 1** *The truth assignment of the variables that is derived from the movement of the literal robots is consistent.*

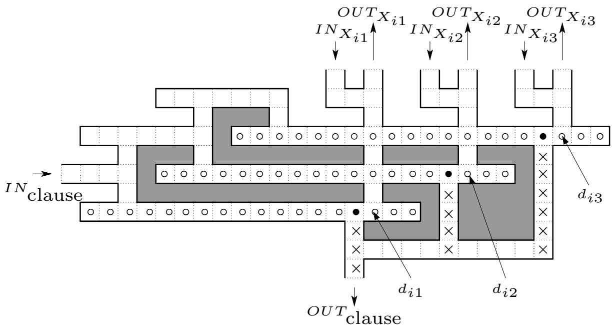


Figure 4: A clause polygon.

For every clause $C_i$ in $\alpha$ we construct a corresponding (sub-)polygon called the *clause polygon* $cp_i$, see

[1]For more details and proofs see our technical report [8], where we discuss also lower bounds on the number of robots needed to reach every cell. For example, it is easy to see that three robots are necessary and sufficient to reach every cell in a rectangular environment.

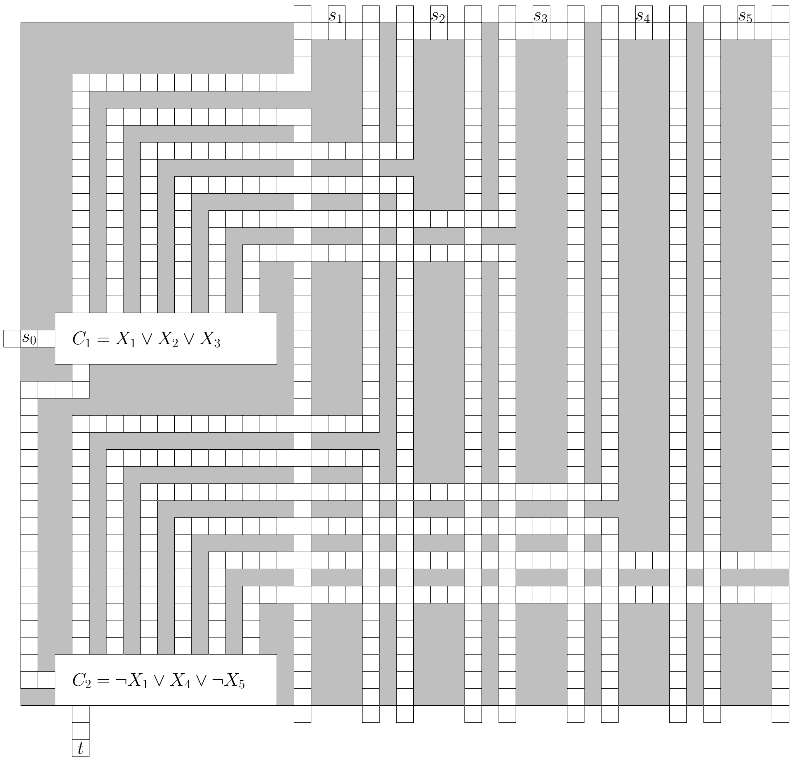

$s_1$
$s_2$
$s_3$
$s_4$
$s_5$
$s_0$
$C_1 = X_1 \vee X_2 \vee X_3$
$C_2 = \neg X_1 \vee X_4 \vee \neg X_5$
$t$

Figure 5: Construction example for $P(\alpha)$ corresponding to $\alpha = C_1 \wedge C_2 = (X_1 \vee X_2 \vee X_3) \wedge (\neg X_1 \vee X_4 \vee \neg X_5)$.

Figure 4. The robot $r_0$ may pass a clause polygon if and only if the clause polygon is visited by at least one of the *literal robots* corresponding to the literals in $C_i$. This, in turn, is only possible, if the clause $C_i$ in $\alpha$ is fulfilled. More precisely, a robot $r_k$ may enter $cp_i$, if it represents $X_k = 1$ and $X_k$ is a literal in $C_i$, or if it represents $X_k = 0$ and $\neg X_k$ is a literal in $C_i$. Moving south into $cp_i$, a robot $r_k$ stops on a cell $d_{ij}$ marked with an arrow in Figure 4. Now, imagine that the robot $r_0$ moves from $IN_{\text{clause}}$ to the horizontal passage marked with $\circ$ where $r_k$ is positioned. The robot $r_0$ stops in front of $r_k$ and can change its direction to enter the vertical passage marked with $\times$ in Figure 4 leading to $OUT_{\text{clause}}$. Note that $r_0$ cannot pass the clause polygon without further help of other robots. The main property of the clause polygons is the following:

**Lemma 2** *The robot $r_0$ may pass a clause polygon from $IN_{\text{clause}}$ to $OUT_{\text{clause}}$ if and only if the corresponding clause in $\alpha$ can be fulfilled.*

In the last step of the construction, we arrange and combine the clause and fork polygons to one polygon $P(\alpha)$: We arrange the clause polygons one beneath the other on the left-hand side of $P(\alpha)$ and the fork polygons side by side on top of $P(\alpha)$, each of them with sufficient space for the connections, see the example in Figure 5. Then we consecutively connect all clause polygons by a passage. More precisely, for $1 \le i < m$ we connect $OUT_{\text{clause}}$ of $cp_i$ and to $IN_{\text{clause}}$ of $cp_{i+1}$. Further, we connect $IN_{\text{clause}}$ of $cp_1$ to the start cell $s_0$ of $r_0$ and $OUT_{\text{clause}}$ of $cp_m$ to the target cell $t$. Thus, $r_0$ has to pass consecutively all clause polygons.

Now we connect the fork polygons to the clause polygons: First, we extend the $X_k$-passages and the $\neg X_k$-passages of the fork polygons to the south until they reach the last clause polygon. Thus, we have $2n$ vertical corridors parallel to the column of clause polygons. Each of these corridors ends in a blind alley. Then we add connections from this bus structure to the clause polygons: If $X_k$ is the $j$th literal in the clause $C_i$ of $\alpha$, we divert $r_k$ from the $X_k$-passage via $IN_{X_{ij}}$ through $cp_i$ and via $OUT_{X_{ij}}$ back to the $X_k$-passage. Remark that we add an obstacle cell to the $X_k$-passage between the horizontal connections to the clause polygon. Analogously, if $\neg X_k$ is the $j$th literal, we connect $IN_{X_{ij}}$ and $OUT_{X_{ij}}$ to the $\neg X_k$-passage. Note that the passages are mostly separated by obstacles—the only exceptions are crossings of connecting passages. But these crossings do not matter, a literal robot $r_k$ stays in its $X_k$- or $\neg X_k$-passages, after it left the fork polygon. Further, $r_0$ cannot reach one of these passages.

Altogether, we arrived at the basic idea of our proof: If $t$ is reachable by $r_0$, $r_0$ must pass every clause polygon. At the same time $r_0$ reaches a clause polygon

$cp_i$, at least one literal robot $r_k$ must enter $cp_i$. This corresponds to the conditions to fulfill $\alpha$: The truth assignment derived from the literal robots that enter the clause polygons ensures that there is at least one literal fulfilled in every clause. On the other hand, any given truth assignment that satisfies $\alpha$ fulfills at least one literal for every clause of $\alpha$; thus, for every clause polygon in $P(\alpha)$ there is at least one literal robot able to enter it. Further, all clauses are connected in a serial way, which corresponds to the conjunction of clauses in $\alpha$. Thus, we can state:

**Lemma 3** *There is a truth assignment that fulfills $\alpha$, if and only if $t$ is reachable by $R$ in $P(\alpha)$.*

It is easy to see that we need a construction time in $O(mn)$, but this time is still polynomial in the size of $\alpha$. Further, we can construct a nondeterministic Turing machine that chooses nondeterministically a sequence of movements to reach $t$ and verifies the reachability of $t$ in polynomial time, see [7] for more details. Altogether, we have:

**Theorem 4** *The Reachability Problem is NP-complete.*

## 4 Summary

We considered robot swarms moving in cellular environments under a new type of movement constraint and showed that the question, whether a cell can be reached by a robot, is NP-complete for arbitrary environments. A Java applet for simulating robot swarms moving under our constraints can be found in

`http://www.geometrylab.de/RacingRobots/`

Interesting open problems are, whether there is a constant lower bound for polygons without holes and without corridors of width 1, and whether there is an efficient algorithm for the Reachability Problem in this type of polygons.

## References

[1] S. Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32:123–143, 2002.

[2] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell, and M. Skutella. The freeze-tag problem: how to wake up a swarm of robots. In *Proc. 13th Annu. ACM-SIAM Symp. Disc. Algor.*, pages 568–577, 2002.

[3] P. Berman. On-line searching and navigation. In A. Fiat and G. Woeginger, editors, *Competitive Analysis of Algorithms.* Springer-Verlag, 1998.

[4] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2–3):231–254, 1995.

[5] A. M. Bruckstein, M. Lindenbaum, and I. A. Wagner. Distributed covering by ant-robots using evaporating traces. *IEEE Trans. Robot. Autom.*, 15:918–933, 1999.

[6] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Boston, 2005.

[7] B. Engels. Navigation in Gitterumgebungen für verteilte Robotersysteme mit eingeschränkter Sensorik. Diplomarbeit, Universität Bonn, August 2005. http://www.geometrylab.de/RacingRobots/.

[8] B. Engels and T. Kamphans. Randolphs robot game is NP-complete! Technical Report 005, Department of Computer Science I, University of Bonn, 2005. http://web.informatik.uni-bonn.de/I/publications/ek-rrgin-05.pdf.

[9] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24:197–224, 2003.

[10] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring an unknown cellular environment. In *Abstracts 16th European Workshop Comput. Geom.*, pages 140–143. Ben-Gurion University of the Negev, 2000.

[11] 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.

[12] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring simple grid polygons. In *11th Internat. Comput. Combin. Conf.*, volume 3595 of *Lecture Notes Comput. Sci.*, pages 524–533. Springer, 2005.

[13] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.

[14] T. Kamphans. *Models and Algorithms for Online Exploration and Search.* PhD thesis, University of Bonn, to appear.

[15] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

[16] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[17] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 116–121, 1985.

[18] A. Randolph. Ricochet robots. Board Game, german edition by Abacus Games, Dreieich, 1999.

[19] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.

# Visibility Map determination using Angle preprocessing

L. Ortega, A.J. Rueda and F. Feito[*]

## Abstract

Visibility determination becomes especially significant in Computer Graphics for walkthrough applications. The aim is to determine those visible objects of the scene from the viewer position. Even when this problem is nowadays efficiently solved by the graphics hardware, when a real-time response is required, some additional CPU processing in the scene geometry may be necessary to deal with large scenes or with observers moving into the scene.

The present work introduces a new technique based on angle preprocessing with *polar diagrams*. The result is displaying $2.5D$ scenes in a more efficient way than using acceleration graphics hardware.

## 1 Introduction

The increasing capabilities of the graphics hardware allows to speed up image generation, obtaining a great realism sensation. However, scene sizes can be increased to become too high to provide interactive frame-rates. A previous process to the rendering phase should be able to discard as much non-visible objects as possible, in order to release the hardware of displaying scene primitives that are invisible from a certain viewer position. These techniques are called *Visibility culling* and some of them have been already incorporated in the hardware. The aim is reducing as much as possible the resulting Potentially Visible Set (PVS) for a high performance in the rendering, obtaining the *visibility map* when the PVS becomes minimum. Occlusion culling methods, summarized in [1, 2, 7], try to reduce the PVS as possible, but not forgetting any visible primitive. The strategies including this characteristic are called *conservative methods*.

Normally, the type of scene if determinant in order to select a proper occlusion culling method. Urban scenes, as well as architectural environments, are considered a special case of densely occluded environments, studied by *cell-portal culling* techniques [8]. If cities consist of a set of 2.5D objects representing buildings instead of 3D models [3, 11], the scene geometry becomes more straightforward to process, what could make possible a exact visibility approach in real time, even if virtual observers are moving into the scene. The additional CPU time to compute the visibility set in the $2.5D$ scene is compensated in the rendering phase.

In this work we propose a new visibility culling approach based on *polar diagrams*, a plane tessellation, that used as preprocessing, solves efficiently some angle-based problems, as Visibility Determination or Convex Hull of points and objects [4, 5], Path Plannig [6] and Collision Detection [10].

Section 2 defines the polar diagram and its angular characteristics. Section 3 studies visibility with polar diagrams as a previous step to find the visible set of objects from any point in the scene (Section 4), and finally Section 5 shows the experimental results when this algorithm is used to display urban scenes.

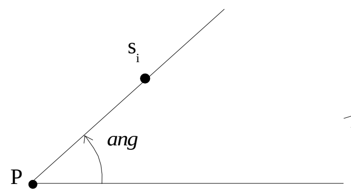
## 2 The polar diagram

The polar diagram of the scene $E$, consisting of $n$ two-dimensional objects, $E = \{o_0, o_1, ..., o_{n-1}\}$, denoted as $\mathcal{P}(E)$, is a plane partition in *polar regions*. Each generator object $o_i$ creates a polar region $\mathcal{P}_E(o_i)$ representing the locus of points with common angular characteristics in a starting direction. Any point in the plane can only belong to a polar region, what determines its angular situation with respect to the rest of generator objects in the scene. More specifically, if point $p$ lies in the polar region of object $o_i$, $p \in \mathcal{P}_E(o_i)$, we know that $o_i$ is the first object found after performing an angular scanning from the horizontal line crossing $p$ in counterclockwise direction [9].

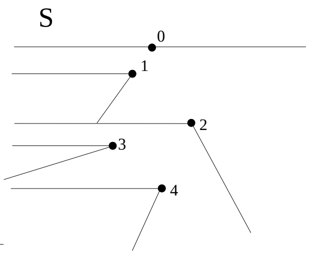
The *polar angle* of point $p$ with respect to $o_i$, denoted as $ang_{o_i}(p)$, is the angle formed by the positive horizontal line of $p$ and the straight line linking $p$ and $o$, as shown in Figure 1.a). The polar angle must be lower than $\pi$, involving that $p$ must be under the horizontal line defined by $s_i$. The locus of points with smaller positive polar angle with respect to $o_i \in E$ is the polar region of $o_i$. Thus, $\mathcal{P}_E(o_i) = \{(x, y) \in E^2 \mid ang_{o_i}(x, y) < ang_{o_j}(x, y), \forall j \neq i\}$. The same criterion is valid for points or any other geometric object. Figure 1.b) depicts the polar diagram of the cloud of points $S$

The polar diagram can be computed efficiently using the Divide and Conquer or the Incremental methods, both working in $\Theta(n \log n)$. The strength of using this tessellation as preprocessing is avoiding any angular sweep by locating a point into a polar region in logarithmic time [4, 5].

[*]Department of Computer Science, University of Jaén, Spain {lidia|ajrueda|ffeito}@ujaen.es



(a) polar angle of $p$ with respect to $s_1$



(b) polar diagram of $S$

Figure 1: Example of polar angle and polar diagram.

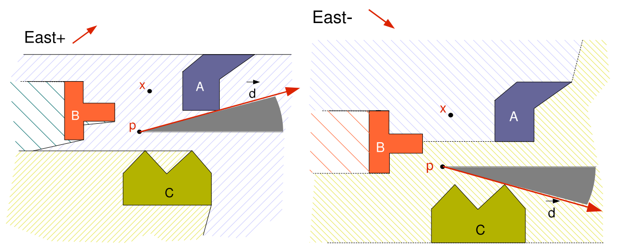


Figure 2: $p \in \mathcal{P}_E(A)$ in the *East*+ polar diagram and $p \in \mathcal{P}_E(C)$ in the *East*$-$ polar diagram.

## 3 Visibility resolution

Polar diagrams can solve Visibility problems due to its capability of determining the nearest angular neighbours. Observe Figure 2 and suppose that the maximum visibility angle from point $p$ in East direction is required. Instead of solving the problem in linear time by performing two angular sweeps, object $A$ is found because point $p$ is located in its polar region in $O(n \log n)$ time, as depicted in Figure 2.a). However, a new polar angle criterion should be necessary to find object $C$, and consequently a new plane tessellation. With the positive polar angle criterion (counterclockwise) the *East*+ polar diagram is constructed, and using the negative one (clockwise), the *East*$-$, as Figure 2.b) shows. The maximum visibility angle is obtained throwing tangent lines towards $A$ in *East*+, and towards $C$ in the *East*$-$ polar diagram.

Summarizing, given the observer position $p$ looking at the East direction, and the pair of polar diagrams *East*+ and *East*$-$, the visibility problem solution can be dealt with a simple result:

- if the point lies in regions associated to different objects in both polar diagrams, as point $p$ in the figure, it always means that there is an open visibility angle in *East*+ direction.
- when a point lies in regions belonging to the same object, the visibility angle is null.

| |
|---|
| if $0 \leq \overrightarrow{d} \leq \pi/2$ use p.d. East+ |
| if $\pi/2 \leq \overrightarrow{d} \leq \pi$ use p.d. West- |
| if $\pi \leq \overrightarrow{d} \leq 3\pi/2$ use p.d. West+ |
| if $3\pi/2 \leq \overrightarrow{d} \leq 0$ use p.d. East- |

Figure 3: The Polar diagram election depends on the direction.

Point $x$ belongs only to the polar regions of object $A$, what implies that the visibility angle is null. Anyway, this information can be enormously important because we know exactly the first object obstructing a trajectory in the specified direction, what becomes useful in collision detection problems.

To manage visibility problems in any other non-orthogonal direction, a fixed and specific set of polar diagrams can be constructed, each of them covering an angular spectrum, a quadrant of the coordinate system, for instance. The *East+* polar diagram can avoid angular sweeps in the interval $[0, \pi/2]$, what means that for all rectilinear movements in this angular interval, the use of the *East* polar diagram is enough to solve visibility problems. If the movement direction is in the range $[\pi/2, \pi]$, the visibility information can be given by the *West-* polar diagram, that is, starting from $\pi$ and sweeping clockwise. Figure 3 shows a table with this correspondence. In the worst case, the first visibility object in a given direction is solved using four polar diagrams.

Actually, visibility problems need to be solved in angular intervals, the equivalent to the observer vision angle. The previous problem is more a collision detection resolution as described in [10]: a point object is thrown from the position $p$ with direction $\overrightarrow{d}$, describing the ray $r(t) = p + t\,\overrightarrow{d}$. The result is the first object intersecting its trajectory. However, this method is the key to compute the visibility map as we describe next.

## 4 Visibility culling in $2D$ scenes

The visibility culling resolution becomes an exact visibility approach using polar diagrams in two-dimensional scenes because it allows to compute the exact set of visible objects, even the set of visible edges, what becomes specially interesting in scenes whose polygons contain a large number of vertices. This new approach is conservative: obtains an angular-sorted list of primitives and is independent of the scene size, that is, the processing time depends only on the visible set size.

Let be $E$ a two-dimensional scene consisting on a set of polygonal objects, $E = \{o_1, o_2, \ldots, o_n\}$, and an observer located in position $p$ looking at the scene in an angular interval $[\vec{r_L}, \vec{r_R}]$, being $\vec{r_L}$ a ray defin-

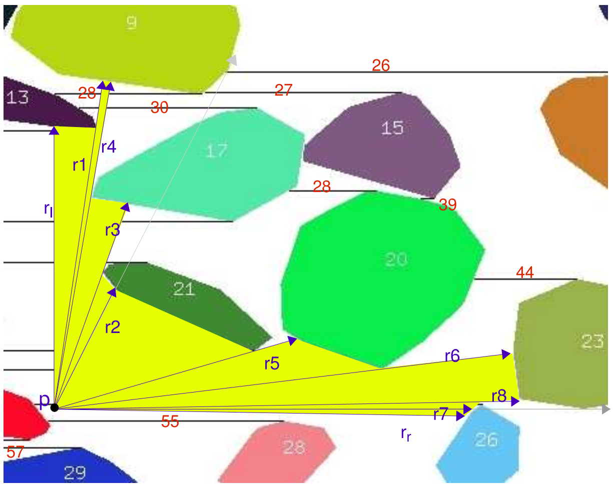


Figure 4: Example of visible set from $p$.

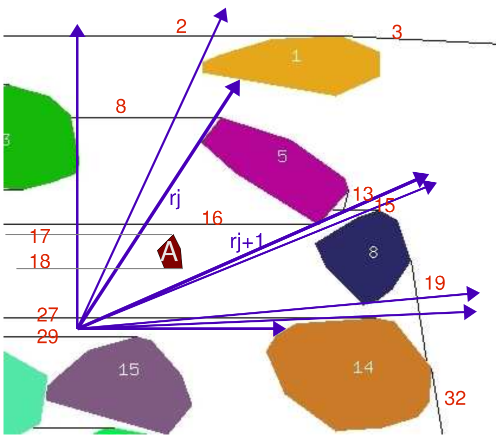


Figure 5: The method is able to find object $A$.

ing the left portion of the angular sector, and $\vec{r_R}$ the one defining the right portion, both rays starting from point $p$ as observed in Figure 4.

- If $\vec{r_L}$ and $\vec{r_R}$ are located in different quadrants, the angular interval $[\vec{r_L}, \vec{r_R}]$ is divided into the resulting sub-intervals from the intersection with the coordinate axes.
- Each of these sub-intervals, $[\vec{r_l}, \vec{r_r}] \subseteq [\vec{r_L}, \vec{r_R}]$ is located in only a polar diagram. For all of them: (1) locate $p$ in a polar region using the appropiate polar diagram and (2) obtain a sorted list of rays $R_{lr}$, between $\vec{r_l}$ and $\vec{r_r}$, $R_{lr} = \{\vec{r_l}, \vec{r_1}, \vec{r_2}, ..., \vec{r_r}\}$, being $\vec{r_j}$, $j \in [i, 1, 2, ..., d]$ a ray starting from point $p$.

For every $r_j$ a collision detection is performed, determining not only the intersected object (or primitive), $o_i$, $i \in [0..N-1]$, but also a set of crossed polar regions, or what becomes similar, the set of crossed polar edges. Let be $l_j$ the sorted set of crossed polar edges by $r_j$ until a collision detection is detected, or until the ray left the scene, $l_j = \{e_k, e_{k+1}, ..., e_m\}$, $k \geq m$. The final set of rays $R_{lr}$, represents the visibility map because each pair $[r_j, r_{j+1}]$, $j \in [l, 1, 2, ..., r-1]$, symbolizes an angular sector (a triangle) from the viewer position $p$ towards an only polygon $o_i$, as depicted in Figure 4.

The method starts from the rays $\vec{r_l}$ and $\vec{r_r}$, the ones defining the vision angle. The rest of rays of $R_{lr}$ are obtained throwing tangent rays towards the involved objects of the collision method described in [10]. Whenever a ray $\vec{r_j}$ reaches a new object $o_i$, the following step consists of throwing the ray $r_{j+1}$ as right tangent (or left tangent depending on the running sequence of the algorithm) from $p$ to $o_i$. The final goal is joining $\vec{r_l}$ and $\vec{r_r}$ with a fan of rays. The ray $\vec{r_j}$ can:

- reach the right (or left) tangent vertex of $o_i$ and then collides with some other object $o_m$ (in the example $\vec{r_1}$ is tangent to $o_{13}$ and intersects with $o_9$).
- not reach the right (or left) tangent vertex of $o_i$ because it collides previously with some other $o_k$ (in the example the ray $\vec{r_2}$, the right tangent line to $o_9$, collides before with $o_{21}$). These cases are easily detected checking the length of the lists of rays; if $l_i$ is longer than $l_j$, as in this example, the search is diverged in two angular intervals, the first between $\vec{r_i}$ and $\vec{r_j}$, and the second between $\vec{r_j}$ and $\vec{r_d}$, that is, $[\vec{r_l}, ..., \vec{r_j}, ..., \vec{r_r}]$. The process starts from $\vec{r_j}$ left to reach $\vec{r_l}$, and right to reach $\vec{r_r}$ recursively, closing the angular sequence.

Conclusions after applying polar diagrams in the described visibility culling technique are the following: $O(2k)$ rays thrown for $k$ visible objects, being $k$ independent of the scene size $n$. In the example ten rays are necessary to locate seven visible objects. The method is conservative because it is able to find all visible objects from $p$ without forgetting any of them. In fact, it is an exact visibility culling method for $2D$ scenes because the exact set of visible edges from $p$ is found.

The example of Figure 5 clearly shows that the method is conservative. Suppose that object $A$ is located in such a position that the described algorithm throws tangent rays, $\vec{r_j}$ and $\vec{r_{j+1}}$ towards tangent positions of object $o_5$, but object $A$ is not reached at all. However $A$ can be detected by checking the lists $l_j = \{e_{27}, e_{18}, e_{17}, e_{16}, e_8, e_2\}$ and $l_{j+1} = \{e_{27}, e_{16}, e_8, e_2\}$. Both begin and end with the same sequence but edges $e_{18}$ and $e_{17}$ does not appear in list $l_{j+1}$, what means that there is another object just in the middle of the angular interval $[\vec{r_j}, \vec{r_{j+1}}]$.

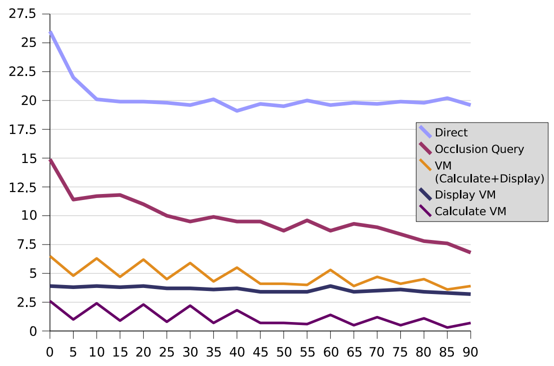


Figure 6: Experimental results.

## 5 Experimental results

Normally the efficiently of occlusion culling methods are highly dependent of the environment characteristics. However, the construction of this plane partition is not altered by the scene density, or by the convexity or non-convexity of occluders, only the number of vertices is relevant. In any case polar diagrams are calculated only once in the preprocessing phase. The location and collision processes are determined by the number of objects $N$. In the calculation of the visibility map, the more visible primitives, the more collision detections are necessary. If occluders are large objects, their capacity to hide some others is greater, and the number of rays to throw probably lower. Furthermore, a collision detection process is dependent on the number of crossed polar edges, and consequently on the number of scene objects.

The experimentation has been carried out in a Pentium IV running at 2.4GHz with a plane city of $N = 600$ buildings with non-convex floor, each of them representing a ground plant of a building or a block of buildings. The total number of vertices in the scene is $n = 45.000$, a number only relevant in the polar diagram construction process. Each of the four polar diagrams are calculated in $100ms$. The rest of algorithms, location or collision detection, run in proportional times to $2N = 12000$ polar edges.

Figure 6 shows the requiring time to display the same view of the city while an observer moves along of part of its outskirts. The required time to display the scene at every step is represented in edge $Y$, while the edge $X$ represents each of this steps. For simplicity, we suppose a vision angle of 90, centered in 45.

The chart is eloquent at all: the top most line represents the time to display the scene directly (*Direct*), without using any CPU or hardware improvement. When hardware acceleration is activated the visualization process is clearly improved (*Occlusion Query* [1]). However using the visibility map and displaying only those visible objects, the time behaviour is still better. The time required for performing the visibility map is appreciably lower that any of the required for visualization, what proves the advantage of using a CPU preprocessing. During the walkthrow process, sometimes the walker sees only a reduced set of building and in other occasions can see the row of buildinds of an avenue. That is the reason of the zigzag observed in the line defining the visibility map (*Visibility Map*). The line labeled as *MV+Display* adds the time required for the visibility map calculation and the one for displaying the resulting visible set.

## References

[1] T. Akenine-Möller and E. Haines. *Real-Time Rendering.* A. K. Peters, 2002.

[2] D. Cohen-Or, Y. Chrysanthou, C.T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transation and Computer Graphics*, 19(3):412–431, Jul-September 2003.

[3] Laura Downs, Tomas Möller, and Carlo H. Séquin. Occlusion horizons for driving through urban scenery. In *Symposium on Interactive 3D Graphics*, pages 121–124, 2001.

[4] C. I. Grima, A. Máquez, and L. Ortega. A locus approach to angle problems in computational geometry. In *Proc. of 14th European Workshop in Computational Geometry*, Barcelona, 1998.

[5] C. I. Grima, A. Máquez, and L. Ortega. Polar diagrams of geometric objects. In *Proc. of 14th European Workshop in Computational Geometry*, pages 149–151, Sophia-Antipolis, 1999.

[6] C.I. Grima, A. Márquez, and L. Ortega. Motion planning and visibility problems using the polar diagram. In *EUROGRAPHICS'03 Short Presentations*, pages 13–19, 2003.

[7] H. Hey and W. Purgathofer. Occlusion culling methods. state of the art report. In *EUROGRAPHICS'01*, Manchester, 2001.

[8] D.P. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potencially visible sets. In *Proceedings 1995 Symposium on Interactive 3D Graphics*, 1995.

[9] L. Ortega. *El Diagrama Polar, Ph Thesis.* University of Seville (Spain), 2002.

[10] Lidia Ortega and Francisco F. Feito. Collision detection using polar diagrams. *Computer & Graphics*, 29(5):726–737, 2005.

[11] P. Wonka, M. Wimmer, and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. In *Proceedings of EUROGRAPHICS*, pages 51–60, 1999.

# Maximizing the Guarded Interior of an Art Gallery[*]

Ioannis Z. Emiris[†] Christodoulos Fragoudakis[‡] Euripides Markou[§]

**Abstract**

In the Art Gallery problem a polygon is given and the goal is to place as few guards as possible so that the entire area of the polygon is covered. We address a closely related problem: how to place a fixed number of guards on the vertices or the edges of a simple polygon so that the total guarded area inside the polygon is maximized. Recall that an optimization problem is called APX-hard, if there exists an $\epsilon > 0$ such that an approximation ratio of $1 + \epsilon$ cannot be guaranteed by any polynomial time approximation algorithm, unless $P = NP$. We prove that our problem is APX-hard and we present a polynomial time algorithm achieving constant approximation ratio. Finally we extend our results for the case where the guards are required to cover valued items inside the polygon. The valued items or "treasures" are modeled as simple closed polygons.

## 1 Introduction

In the Art Gallery problem a polygon is given and the goal is to place as few guards as possible so that the entire area of the polygon is covered. Many variations of the Art Gallery problem have been studied during the last two decades ([11], [12], [13]). These can be classified with respect to where the guards are allowed to be placed (e.g. on vertices, edges, interior of the polygon) or whether only the boundary or all of the interior of the polygon needs to be guarded, etc. Most known variations are NP–hard. We address a closely related problem:

**Definition 1** *Given is a simple polygon $P$ and an integer $k > 0$. The goal of the* Maximum Area Vertex Guards *problem is to place $k$ vertex guards so that the area of $P$'s interior that is overseen by the guards is maximum.*

The Minimum Vertex Guards problem asks how to guard a polygon, with or without holes, using a minimum number of guards placed on vertices; extensions consider edges or points in the interior. These problems are APX–hard and O($\log n$)–approximable [8, 3, 4]. A related problem about terrain guarding, is the Minimum Fixed Height Vertex (Point) Guards On Terrain problem ($\Theta(\log n)$–approximable [6], [3], [4]). In [7] the case of guarding the walls (and not necessarily every interior point) is studied. In [2] the following problem has been introduced: suppose we have a number of valuable treasures in a polygon $P$; what is the minimum number of mobile (edge) guards required to patrol $P$ in such a way that each treasure is always visible from at least one guard? In [2] they show NP–hardness and give heuristics for this problem. In [1] weights are assigned to the treasures in the gallery. They study the case of placing one guard in the gallery in such a way that the sum of weights of the visible treasures is maximized. Recent (non- )approximability results for art gallery problems can be found in [8, 11, 12, 13, 4, 6].

## 2 Finest Visibility Subdivision

We recall from [5] and [9] some preliminary definitions: Let $P$ be a simple polygon, $a, b \in P$ two points inside $P$ and $L, M \subseteq P$ two sets of points inside $P$. We say that point $a$ *sees* point $b$, i.e. $a$ and $b$ are mutually visible, if the segment connecting $a$ and $b$ lies inside the closed polygon $P$. We say that the point set $L$ is *visible* from the point set $M$ or that $M$ *oversees* $L$ if for every point $a$ which belongs to $L$, there exists a point $b$ that belongs to $M$, such that $a$ *sees* $b$. Notice that if $M$ oversees $L$, it is not necessary for $L$ to oversee $M$. Finally, $M$ *watches* $L$ if there exists a point $a$ that belongs to $L$ and a point $b$ that belongs to $M$ such that $a$ *sees* $b$. Notice that if $M$ watches $L$ then also $L$ watches $M$.

Our method descritizes the interior of any simple polygon with respect to visibility. In [5] we defined the notion of the *Finest Visibility Segmentation* of the boundary of $P$: Consider the visibility graph $V_G(P)$ with vertex set $V(P)$, i.e. the vertex set of $P$, where two vertices share an edge iff they are visible in $P$. By extending the edges of $V_G(P)$ inside $P$ up to the boundary of $P$ we obtain a set of points $FVS$ of the boundary of $P$, that includes of course all vertices of $P$. An extended edge of $V_G(P)$ generates at most two $FVS$ points. Since there are $O(n^2)$ edges in $V_G(P)$, there are $O(n^2)$ points in any polygon's $FVS$ set. We

[*]Research partially supported by PYTHAGORAS (project 70/3/7392 under the EPEAEK program of the Greek Ministry of Educational Affairs.

[†]Department of Informatics and Telecommunications, National University of Athens, `emiris@di.uoa.gr`

[‡]Computer Science, ECE, National Technical University of Athens, `cfrag@cs.ntua.gr`

[§]Department of Informatics and Telecommunications, National University of Athens, `emarkou@di.uoa.gr`

call this construction the *Finest Visibility Segmentation* of the boundary of $P$. Any *open* segment $(a, b)$, (i.e. $a$ and $b$ are excluded), defined by consecutive $FVS$ points, is called an $FVS$ segment of $P$.

Here we extend the $FVS$ construction considering also all the intersection points of all the extended visibility graph's edges inside $P$. There are $O(n^4)$ regions created inside $P$ which are called $FVS$ regions of $P$. Due to the above construction, such an $FVS$ region has the following property: none of the extended visibility edges can cross the region. We call this construction the *Finest Visibility Subdivision* of the interior of $P$.

The following two lemmas establish that a $FVS$ region cannot be **only partly** visible from a vertex or an edge.

**Lemma 1** *For any vertex $v$ of a simple polygon $P$, a $FVS$ region is visible by $v$ if and only if it is watched by $v$.*

**Proof.** Of course if a $FVS$ region is visible by $v$, then it is watched by $v$. Suppose now that the region is watched by $v$ but not overseen by $v$. In that case there must be another vertex $v'$ which blocks the visibility of $v$. But then $vv'$ is a visibility edge and the extension of $vv'$ crosses the $FVS$ region which cannot hold since it contradicts the definition of a $FVS$ region. □

**Lemma 2** *For any edge $e$ of $P$, a $FVS$ region is visible by $e$ if and only if it is watched by $e$.*

**Proof.** (**Sketch**) Of course if a $FVS$ region is visible by $e$, then it is watched by $e$. Suppose now that the region is watched by $e = (v_i, v_j)$ but not overseen by $e$. This means that there is a point $p$ inside the $FVS$ region which sees a point $d \in e$. Sweeping the line that passes through $p$ and $d$ around $d$ we meet a vertex $v_m$ of $P$ before the sweep lines stops crossing the $FVS$ region (since the region is not overseen). We start now sweeping the line that passes through $d$ and $v_m$ around $v_m$ until the line stops crossing the $FVS$ region. Notice that if another vertex $v_n$ is hit by the sweep line then there must be a visibility edge (e.g. $v_m v_n$) whose extension crosses the $FVS$ region. But the latter cannot hold since by definition an $FVS$ region is not crossed by any visibility edges. □

The above lemmas demonstrate the following: The interior of $P$ can be effectively descritized in terms of visibility to $O(n^4)$ $FVS$ regions. Any vertex (edge) of $P$ oversees a $FVS$ region if and only if it watches the $FVS$ region. In order to find the set of all overseen $FVS$ regions from a polygon vertex $v$, namely the $FVS(v)$ set, (using lemma 1) it suffices to select a point $p$ inside every $FVS$ region and connect it to vertex $v$. If this segment is everywhere inside the polygon $P$, then the region containing $p$ is overseen from $v$.

For the case of the $FVS(e)$ set, that is the set of all overseen $FVS$ segments from a polygon edge $e$, (using lemma 2) it suffices to select a point $p$ inside every $FVS$ region and then sweeping a line around $p$. The $FVS$ region containing $p$ is overseen by an edge touched by that line.

## 3 The Maximum Area Vertex (Edge) Guards problem

Let $A(r)$ be the area of region $r$.

**Algorithm 1** Maximum Area Vertex Guards

```
compute the $FVS$ regions
for all $v \in V(P)$ do
  compute $FVS(v)$
end for
$SOL \leftarrow \emptyset$
for $i = 1$ to $k$ do
  select $v \in V$ that maximizes $A(SOL \cup FVS(v))$
  $SOL \leftarrow SOL \cup FVS(v)$
end for
return $A(SOL)$
```

Consider $P$ and integers $k, A > 0$. It is NP–hard to decide whether we can place at most $k$ guards on vertices of $P$ so that the total area overseen by the guards is at least $A$. To see why, consider the decision Minimum Vertex Guards problem, which asks whether the interior is overseen by at most $k$ guards. The reduction to Maximum Area Vertex Guards is straightforward.

In [10] they prove that to maximize the guarded boundary of a polygon with or without holes using guards placed on vertices or edges, is APX–hard. They present a gap preserving reduction from Max-5-Occurence-3-SAT to Maximum Value Vertex Guard problem. In the construction part, methods from [8] for NP-hardness and [3] for APX-hardness are combined. If we change the construction part of the reduction so that to make sure that the area touched by the (so called) "cheap edges" is small enough then the following theorem holds:

**Theorem 3** *The* Maximum Area Vertex (Edge) Guards *problem is APX–hard.*

Alg. 1 is an approximation algorithm for the Maximum Area Vertex Guards problem. It starts by calculating the $FVS$ regions and then for every $v \in V(P)$ the set $FVS(v)$. During each iteration of the algorithm, for any vertex $v$ that hasn't been assigned a guard yet, the set $SOL \cup FVS(v)$ (of the overseen regions) is found and its area is calculated. The vertex that maximizes the total area of the (not previously) overseen regions is then chosen, causing a maximum possible increase of the solution. Then the algorithm updates the set $SOL$ by adding the new $FVS$ regions.

In order to prove that algorithm 1 approximates MAXIMUM AREA VERTEX GUARDS by a constant approximation factor, we work as follows:

Let $OPT$ denote the collection of the set of regions in an optimal solution and $SOL$ denote the collection returned by the algorithm. These collections have $A(OPT)$ and $A(SOL)$ values respectively. Suppose that the algorithm places a guard at vertex $v_i$ at iteration $i$, and a set of new regions $P_i$. Therefore the added total value of regions at iteration $i$ is $A(P_i)$.

Consider the ordered sequence of vertices (as they have been selected by the algorithm) and let $v_l$ be the first vertex in the sequence where a guard has been placed by the algorithm but not in the optimal solution. It holds:

$$A(P_i) = A(\cup_{m=1}^{i} P_m) - A(\cup_{m=1}^{i-1} P_m)$$

**Lemma 4** *After $l$ iterations of algorithm 1, we have, for $l = 1, 2, \dots, k$,*

$$A(\cup_{i=1}^{l} P_i) - A(\cup_{i=1}^{l-1} P_i) \geq \frac{A(OPT) - A(\cup_{i=1}^{l-1} P_i)}{k}.$$

**Proof.** Consider vertices where guards have been placed in the optimal solution but no guard has been placed there by the algorithm. By the pigeonhole principle, there is at least one such vertex $v_m$ so that the following holds:

$$A(P'_m) \geq \frac{A(OPT) - A(\cup_{i=1}^{l-1} P'_i)}{k} \Rightarrow$$

$$\Rightarrow A(P'_m) \geq \frac{A(OPT) - A(\cup_{i=1}^{l-1} P_i)}{k}.$$

Notice that $A(P_l) \geq A(P_m) \geq A(P'_m)$ and $A(\cup_{i=1}^{l} P_i) - A(\cup_{i=1}^{l-1} P_i) = A(P_l)$, therefore:

$$A(\cup_{i=1}^{l} P_i) - A(\cup_{i=1}^{l-1} P_i) \geq \frac{A(OPT) - A(\cup_{i=1}^{l-1} P_i)}{k}$$

□

**Lemma 5** *After $l$ iterations of algorithm 1 it holds:*

$$A(\cup_{i=1}^{l} P_i) \geq (1 - (1 - \frac{1}{k})^{l}) A(OPT), \ \ l = 1, ..., k$$

**Proof.** We are going to prove this by induction on $l$. During the first step of the algorithm the set with value $A(P_1)$ is chosen. It holds:

$$A(P_1) \geq A(P'_1)$$

$A(P'_1)$ is the maximum possible value that $OPT$ achieves, so from the pigeonhole principle:

$$A(P'_1) \geq \frac{A(OPT)}{k} \rightarrow A(P_1) \geq \frac{A(OPT)}{k}$$

Suppose that the relation holds for $i = l - 1$:

$$A(\cup_{i=1}^{l-1} P_i) \geq (1 - (1 - \frac{1}{k})^{l-1}) A(OPT).$$

Since $A(\cup_{i=1}^{l} P_i) = A(\cup_{i=1}^{l-1} P_i) + (A(\cup_{i=1}^{l} P_i) - A(\cup_{i=1}^{l-1} P_i))$, using lem. 4 we have:

$$A(\cup_{i=1}^{l} P_i) \geq A(\cup_{i=1}^{l-1} P_i) + \frac{A(OPT) - A(\cup_{i=1}^{l-1} P_i)}{k} \rightarrow$$

$$A(\cup_{i=1}^{l} P_i) \geq A(\cup_{i=1}^{l-1} P_i)(1 - \frac{1}{k}) + \frac{A(OPT)}{k}$$

From the inductive hypothesis:

$$A(\cup_{i=1}^{l} P_i) \geq (1 - (1 - \frac{1}{k})^{l-1}) A(OPT)(1 - \frac{1}{k}) + \frac{A(OPT)}{k}$$

$$\rightarrow A(\cup_{i=1}^{l} P_i) \geq (1 - (1 - \frac{1}{k})^{l}) A(OPT)$$

□

**Theorem 6** *Algorithm 1 runs in polynomial time and achieves an approximation ratio of $\frac{1}{1-\frac{1}{e}} \approxeq 1.58$ for the* MAXIMUM AREA VERTEX GUARDS *problem.*

**Proof.** Using lemma 5, we set $l = k$ and get:

$$A(\cup_{i=1}^{k} P_i) \geq (1 - (1 - \frac{1}{k})^{k}) A(OPT)$$

It holds:

$$\lim_{k \to \infty} (1 - (1 - \frac{1}{k})^{k}) = 1 - \frac{1}{e}$$

As $(1 - (1 - \frac{1}{k})^{k})$ continuously gets smaller, we have:

$$1 - (1 - \frac{1}{k})^{k} \geq 1 - \frac{1}{e}$$

So:

$$A(SOL) > (1 - \frac{1}{e}) W(OPT)$$

That is the algorithm approximates the MAXIMUM AREA VERTEX GUARDS problem with a $\frac{1}{1-\frac{1}{e}} \rightarrow 1.58$ ratio. □

The algorithm's complexity is $O(n^4)$ because this is the size of $FVS$. A different approach, based on a triangulation and avoiding the $FVS$, yields $O(n^2k^2)$ for the same algorithm.

Similarly as before the MAXIMUM AREA EDGE GUARDS problem is APX–hard. A similar algorithm to algorithm 1 approximates the MAXIMUM AREA EDGE GUARDS problem. The only difference from algorithm 1 is that we need to calculate the $FVS(e)$ set using the technique described in section 2.

All algorithms apply when the polygons have holes.

## 4 The MAXIMUM TREASURES VALUE VERTEX (EDGE) GUARDS problem

Consider the following problem: There exists a simple polygon $P$ which encloses a number of simple subpolygons each of which has a non-negative value assigned. The MAXIMUM TREASURES VALUE VERTEX (EDGE) GUARDS problem's goal is to place vertex or edge guards in a way which maximizes the total value of the overseen or watched subpolygons. To show the usefulness of $FVS$, we prove the following: Given is $P$, which encloses a number of simple subpolygons, each of which has a value assigned. Given also are two integers $k, M > 0$. It is NP–hard to decide whether we can place at most $k$ vertex or edge guards so that the total value of the overseen or watched subpolygons is at least $M$. The proof goes as follows: The decision version of MINIMUM VERTEX GUARDS problem for $P$ reduces to the corresponding decision version of the MAXIMUM TREASURES VALUE VERTEX (EDGE) GUARDS problem for the same polygon: Construct all $FVS$ regions of $P$ and assign value 1 to each region; take as $M$ the total number of $FVS$ regions. Now the reduction is staightforward: The polygon's interior is overseen by at most $k$ guards if and only if the total value of the overseen $FVS$ regions is at least $M$.

A stronger negative result is the following:

**Theorem 7** MAXIMUM TREASURES VALUE VERTEX (EDGE) GUARDS *is APX–hard.*

For this proof, we can change the construction part of the reduction presented in [10] by adding small enough subpolygons touching the "cheap edges". Then, we assign a suitable small value to each subpolygon.

Alg. 1, with the appropriate modifications, approximates also the MAXIMUM TREASURES VALUE VERTEX (EDGE) GUARDS problem with the same ratio as in theorem 6. In fact, the computation of the $FVS$ regions is not required for the case of vertex watching guards, since we can easily compute the subpolygons that are watched by a vertex guard. However, for the case of edge guards, a subpolygon $p_i$ is watched by an edge $e$ if and only if there is a $FVS$ region watched by $e$ that touches $p_i$. The total value of subpolygons in $A(SOL \cup FVS(v))$ is also required.

Notice that all the algorithms can be applied even when the polygons have holes.

## 5 Open problems

Interesting problems are the following: (a) How to place guards and given subpolygons in the polygon so that a maximum value is guarded (i.e. this time we need to place also the subpolygons). (b) How to place guards in the interior of $P$ for all the above problems.

## References

[1] Carlsson S. and Jonsson H. Guarding a Treasury. In *Proc. 5th Canadian Conf. on Computational Geometry*, pages 85–90, 1993.

[2] Deneen L. and Joshi S. Treasures in an Art Gallery. In *Proc. 4th Canadian Conf. on Computational Geometry*, pages 17–22, 1992.

[3] Eidenbenz S. Inapproximability Results for Guarding Polygons without Holes. In *ISAAC*, volume 1533 of *LNCS*, pages 427–436, 1998.

[4] Eidenbenz S. *(In–)Approximability of Visibility Problems on Polygons and Terrains.* PhD thesis, ETH Zurich, 2000.

[5] Fragoudakis C., Markou E., and Zachos S. How to Place Efficiently Guards and Paintings in an Art Gallery. In *10th Panhellenic Conference on Informatics*, volume 3746 of *LNCS*, pages 145–154, 2005.

[6] Ghosh S. Approximation Algorithms for Art Gallery Problems. In *Proc. Canadian Information Processing Society Congress*, pages 429–434, 1987.

[7] Laurentini A. Guarding the Walls of an Art Gallery. *The Visual Computer Journal*, 15:265–278, 1999.

[8] Lee D. and Lin A. Computational Complexity of Art Gallery Problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[9] Markou E., Fragoudakis C., and Zachos S. Approximating Visibility Problems within a constant. In *3rd Workshop on Approximation and Randomization Algorithms in Communication Networks*, pages 91–103, 2002.

[10] Markou E., Zachos S., and Fragoudakis C. Maximizing the Guarded Boundary of an Art Gallery is APX–Complete. In *Proc. CIAC*, volume 2653 of *LNCS*, pages 24–35, 2003.

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

[12] Shermer T. Recent Results in Art Galleries. In *Proceedings of the IEEE*, 1992.

[13] Urrutia J. Art Gallery and Illumination Problems. In *Handbook of Computational Geometry.* Elsevier, 2000.

# On Realistic Terrains*

Esther Moet  Marc van Kreveld  A. Frank van der Stappen

**Abstract**

We study worst-case complexities of the visibility map, the shortest path map, and the Voronoi diagram on terrains under realistic assumptions on edge length ratios and the angles of the triangles. We show that their complexities are considerably lower on realistic terrains than in the general case.

## 1 Introduction

One of the main objectives of computational geometry is to uncover the computational complexity of geometric problems. It provides a theory that explains how efficiently geometric problems can be solved that arise in applications. However, in many cases a discrepancy exists between the provable worst-case computational complexity of an algorithm and the actual running time behavior of that algorithm on inputs that arise in applications. This has led to the study of *fatness* and *realistic input models.*

Among the first applications of realistic input models in computational geometry, Alt *et al.* [1] consider motion planning for a rectangular robot. The efficiency depends on the aspect ratio of this rectangle. Matoušek *et al.* [8] show that if all triangles of a set of $n$ triangles have their angles bounded away from zero (at least $\alpha$, for some constant $\alpha > 0$), then the union of these triangles has $O(n)$ holes rather than $O(n^2)$ for the general case, and the boundary complexity of the union is $O(n \log\log n)$. Such triangles are called *fat.* Since then, various definitions of fatness [2, 5, 12, 14] have been proposed. Other realistic models—such as low density [13], unclutteredness [4], and simple-cover complexity [10]—consider the spatial distribution of objects or their features. An overview of reduced combinatorial complexities and improved algorithmic efficiencies for inputs satisfying these models is given in [4] along with a model hierarchy.

Realistic assumptions have not yet been studied for polyhedral terrains. However, several geometric structures on terrains have complexities much higher than typical in applications. For example, the visibility map of a polyhedral terrain of $n$ triangles has complexity $\Theta(n^2)$ in the worst case, a shortest path has $\Theta(n)$ complexity, and even a bisector of two points on a terrain can have quadratic size. Hence, the discrepancy between theoretical complexity bounds and typical complexity bounds exists on terrains as well. In this paper, we analyze this discrepancy by studying realistic assumptions on terrains.

For visibility maps, we give three assumptions whose combination provides an $O(n\sqrt{n})$ bound on the visibility map of a terrain when viewed from infinity, and an $O(n\sqrt{n} \log\log n)$ bound for perspective views. Dropping any of the three assumptions immediately makes an $\Omega(n^2)$ lower bound construction possible. With our three assumptions, we provide a lower bound construction of size $\Omega(n\sqrt{n})$, matching the upper bound for views from infinity. It is interesting to note that the assumptions all refer to the $xy$-projection of the terrain.

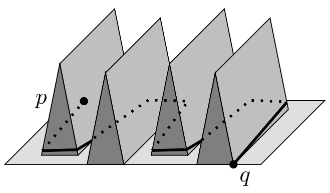


Figure 1: A terrain where in the projection, all its vertices lie on a regular grid and the shortest path between $p$ and $q$ crosses $\Omega(n)$ triangles.

Next, we consider distance structures on terrains. Using only the three assumptions for visibility, we can still have shortest paths that have linear complexity; see Figure 1. Therefore, we introduce a fourth assumption that relates to the steepness of the terrain, and show that any shortest path between two points passes through only $\Theta(\sqrt{n})$ triangles. For a bisector between two points, we show that the same set of four assumptions gives an $O(n\sqrt{n})$ complexity bound rather than quadratic. We give an $\Omega(n)$ size lower bound. The shortest path map for a source point $s$ is the subdivision of the terrain into regions where the combinatorial structure of shortest paths from $s$ is the same. In general it has complexity $\Theta(n^2)$, but we show that under our assumptions it is $\Theta(n\sqrt{n})$. Finally, we study Voronoi diagrams on terrains, which in general also have quadratic complexity, even for only two sites. Our assumptions allow us to prove an upper bound of $O(n\sqrt{n})$, and we give a lower bound of $\Omega(n+m\sqrt{n})$ in case there are $m$ sites on the terrain.

Algorithmically, our results imply faster compu-

*Department of Information and Computing Sciences, Universiteit Utrecht, {esther,marc,frankst}@cs.uu.nl

tation of visibility maps on realistic terrains. The output-sensitive construction of the visibility map of a terrain by Katz *et al.* [7] implies that for realistic terrains, it can be computed in $O(n\sqrt{n}\log n)$ time for views from infinity, and in $O(n\sqrt{n}\log n \log\log n)$ time for perspective views.

Shortest paths on general terrains can be computed in $O(n\log^2 n)$ time [6]. This improved earlier algorithms [3, 9]. The description of Kapoor [6] does not provide sufficient detail to analyze whether a slightly faster algorithm can be obtained for realistic terrains. For shortest path maps and Voronoi diagrams, the algorithm of Mitchell *et al.* [9] leads to $O(n\sqrt{n}\log n)$ time bounds for the construction on realistic terrains.

The proofs and lower bound constructions that we omit in this extended abstract can be found in the full version of this paper [11].

## 2 Realistic model

Let $\mathcal{T}$ be a polyhedral terrain, comprising a set $T$ of $n$ triangles, a set $E$ of $n_e$ edges, and a set $V$ of $n_v$ vertices, where $n_e$ and $n_v$ are $O(n)$. We assume $\mathcal{T}$ satisfies the following three properties:

**1.** the minimum angle of any triangle in $T$ in the projection to the $xy$-plane is at least $\alpha$,

**2.** the boundary of the projection of $\mathcal{T}$ onto the $xy$-plane is a rectangle with side lengths 1 and $c$,

**3.** the longest $xy$-projection over all edges in $E$ is at most $d$ times as long as the shortest one.

The values $\alpha$, $c$, and $d$ in the above assumptions are all positive constants. A projected triangle that satisfies the first assumption is *fat* [8]. We call a polyhedral terrain $\mathcal{T}$ a *realistic terrain* if $\mathcal{T}$ satisfies assumptions 1, 2, and 3. Realistic terrains have certain properties, stated in the lemmas and corollaries below, which we use in the next section to prove upper bounds on the worst-case complexities of visibility structures.

**Lemma 1** *Every edge of a realistic terrain has length* $\Theta\left(\frac{1}{\sqrt{n}}\right)$ *in the projection.*

**Corollary 2** *Every triangle in a realistic terrain has area* $\Theta\left(\frac{1}{n}\right)$ *in the projection.*

**Lemma 3** *Two vertices $v$ and $w$ in a realistic terrain have distance* $\Omega\left(\frac{1}{\sqrt{n}}\right)$ *in the projection.*

**Corollary 4** *Two edges $e$ and $e'$ in a realistic terrain that have no common endpoint have distance* $\Omega\left(\frac{1}{\sqrt{n}}\right)$ *in the projection.*

The terrain in Figure 1 satisfies the three assumptions above, but a shortest path between two points on the terrain still passes through $\Theta(n)$ triangles in the worst case. In Section 4, we discuss distance structures on terrains, and to bound their complexity, we introduce an additional assumption:

**4.** the dihedral angle of the supporting plane of any triangle in $T$ with the $xy$-plane is at most $\beta$, where $\beta < \frac{\pi}{2}$ is some constant.

Assumption 4 implies that the maximum slope of a line segment on any triangle of $\mathcal{T}$ is $\tan\beta = O(1)$. In Section 4, we call a terrain *realistic* if it satisfies all four assumptions.

All complexity bounds for the visibility map of a realistic terrain in Section 3 are achieved with only the first three assumptions. In particular, we do not need a bound on the dihedral angles to prove the upper bounds, and the lower bound constructions are all possible with bounded dihedral angle. In the full version of this paper [11], we show that the first three assumptions are necessary to obtain a subquadratic upper bound on the complexity of the visibility map.

## 3 Complexity of the visibility map

Let $p$ be the viewpoint of the visibility map $\mathrm{VM}(p,\mathcal{T})$ that we consider. We bound the complexity of the visibility map by giving an upper bound on $\#I_t$, the number of terrain triangles with which a given triangle $t$ can *interact*, i.e., with which it can create features of $\mathrm{VM}(p,\mathcal{T})$. Recall that a vertex of the visibility map of $p$ directly corresponds to a line through $p$ that is a common tangent of two terrain edges. Since any two triangles create $\Theta(1)$ vertices of $\mathrm{VM}(p,\mathcal{T})$ in the worst case, we get the following expression:

$$\text{Complexity of } \mathrm{VM}(p,\mathcal{T}) = O\left(\sum_{t\in T}\#I_t\right).$$

We call the locus of the triangles with which a triangle $t$ interacts in the visibility map of $p$ the *influence region* of $t$, and we denote it by $\mathcal{R}_t$. Using this definition, we can bound $\#I_t$ from above by the number of triangles in $T$ whose projection intersects $\mathcal{R}_t$. We distinguish two cases based on the location of the viewpoint: (a) $p$ is located infinitely far away from $\mathcal{T}$ (parallel projection), and (b) $p$ lies on or above $\mathcal{T}$ (perspective projection).

In case (a), for any triangle $t$, all triangles that intersect the influence region $\mathcal{R}_t$ are contained in a slab bounded by two parallel lines, whose width is three times the length of the longest edge in $E$; see Figure 2(a). We give bounds on the complexity of $\mathrm{VM}(p,\mathcal{T})$ in this case in Section 3.1. In case (b), the influence region of a triangle $t$ is a truncated wedge; see Figure 2(b). In Section 3.2, we bound the complexity of $\mathrm{VM}(p,\mathcal{T})$ in this case.

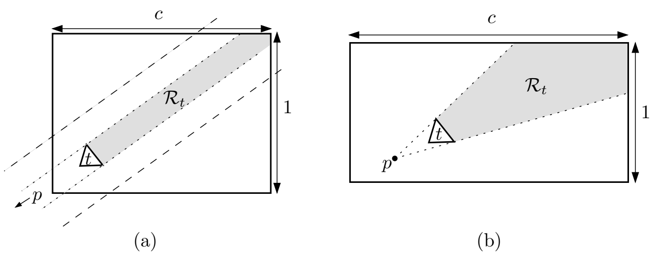


Figure 2: The influence region for a triangle $t$ and (a) a viewpoint at infinity or (b) a viewpoint on or above the terrain.

## 3.1 Viewpoint at infinity

Under the assumptions of Section 2, we can place $\Theta(\sqrt{n})$ triangles in a rectangle of constant length and of width $\Theta(\frac{1}{\sqrt{n}})$. By placing $\Omega(\sqrt{n})$ triangles at one end of this rectangle, each of which interacts with $\Omega(\sqrt{n})$ triangles at the other end, we can get a parallel projection with complexity $\Omega(n)$ for this rectangle.

Since the projection of $\mathcal{T}$ is a rectangle with side lengths 1 and $c$, we can replicate this construction $\Omega(\sqrt{n})$ times, resulting in a visibility map of complexity $\Omega(n\sqrt{n})$ in total; see Figure 3(a).

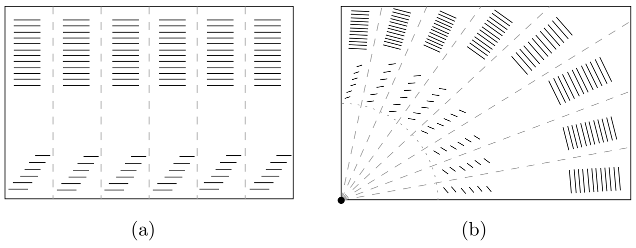


Figure 3: The visibility map of (a) a viewpoint at infinity, or (b) a viewpoint on or above the terrain, can have $\Omega(n\sqrt{n})$ vertices.

**Lemma 5** *The visibility map of a realistic terrain for a viewpoint at infinity can have $\Omega(n\sqrt{n})$ vertices.*

We now show that this bound is tight. Let $l_1$ and $l_2$ be two parallel lines in the $xy$-plane, both intersecting the projection of $\mathcal{T}$, at distance three times the length of the longest edge in $E$. Let $L$ be the slab that is formed by $l_1$ and $l_2$. For every triangle $t$ of $T$ and every viewpoint at infinity, there exists such a slab $L_t$ that completely contains all the triangles that intersect $\mathcal{R}_t$. By Lemma 1, for any triangle $t$ from $T$, the area of $L_t$ is $O(1/\sqrt{n})$, and thus, by Corollary 2, the number of triangles in $L_t$ is $O(\sqrt{n})$. Now, by the discussion at the beginning of Section 3 and Lemma 5, we have the following theorem.

**Theorem 6** *Let $\mathcal{T}$ be a realistic terrain with $n$ triangles, and let $p$ be a viewpoint at infinity. Then* $\mathrm{VM}(p, \mathcal{T})$ *has complexity $\Theta(n\sqrt{n})$ in the worst case.*

## 3.2 Viewpoint on or above the terrain

We can create a subconstruction with $\Omega(\sqrt{n})$ triangles that are located in a wedge of area $\Omega(1/\sqrt{n})$, instead of in a rectangle of the same area, as is the case in Figure 3(a). If we place the viewpoint at the apex of the wedge, then this subconstruction produces a visibility map with $\Omega(n)$ vertices. We can replicate this construction $\Omega(\sqrt{n})$ times in a rectangle of $\Theta(1)$ area; Figure 3(b) displays the construction schematically.

**Lemma 7** *The visibility map of a realistic terrain for a viewpoint on or above the terrain can have $\Omega(n\sqrt{n})$ vertices.*

To obtain an almost matching upper bound, we subdivide the projection of $\mathcal{T}$ into annuli of increasing size and increasing distance from $p$ and bound the number of visibility map vertices for every region separately; this captures the intuition that triangles far away from the viewpoint contribute less to the visibility map than triangles close to the viewpoint. In this way, we can prove the following lemma.

**Lemma 8** *The visibility map of a point on or above a realistic terrain has complexity $O(n\sqrt{n}\log\log n)$.*

Summarizing the results in this section gives us the following theorem.

**Theorem 9** *Let $\mathcal{T}$ be a realistic terrain with $n$ triangles, and let $p$ be a point located on or above $\mathcal{T}$. Then* $\mathrm{VM}(p, \mathcal{T})$ *has worst-case complexity $\Omega(n\sqrt{n})$ and $O(n\sqrt{n}\log\log n)$.*

## 4 Complexity of the shortest path map and the Voronoi diagram

Shortest paths on terrains were studied extensively in [3, 6, 9]. It is easy to see that on any realistic terrain $\mathcal{T}$, two points exist whose shortest path passes through $\Omega(\sqrt{n})$ triangles. We show a matching upper bound. First, we can show that the shortest path between two points on a realistic terrain has constant length, because it cannot be more than a constant times as long as its $xy$-projection. By the results of Section 2, this leads to the following lemma.

**Lemma 10** *Let $\mathcal{T}$ be a realistic terrain with $n$ triangles. The shortest path over $\mathcal{T}$ between two points $p$ and $q$ on $\mathcal{T}$ passes through $\Theta(\sqrt{n})$ triangles in the worst case.*

The bisector $B(p, q)$ of a point $p$ and a point $q$ on $\mathcal{T}$ is the set of points on $\mathcal{T}$ with equal distance to $p$ and $q$. It is a simple curve (open or closed) that consists of line segments and hyperbolic arcs [9]. The worst-case complexity of a bisector is $\Theta(n^2)$ on general terrains. The bisector $B(p, q)$ has $O(n)$ *breakpoints*, which are the points that have two shortest paths to $p$ or two shortest paths to $q$. In [11], we show that on realistic terrains, a bisector intersects $\Theta(\sqrt{n})$ triangles between two breakpoints in the worst case.

**Lemma 11** *For two points $p$ and $q$ on a realistic terrain $\mathcal{T}$, the bisector $B(p,q)$ has complexity $O(n\sqrt{n})$.*

The *shortest path map* of a source point $s$ is the subdivision of $\mathcal{T}$ into cells such that the vertex and edge sequence of the shortest path to any point in that cell from the source is the same. In general terrains, this structure has worst-case complexity $\Theta(n^2)$ [9]. A global analysis of where the boundaries of the shortest path map cells originate from, yields an $O(n\sqrt{n})$ size bound for realistic terrains.

**Theorem 12** *The shortest path map of a realistic terrain has complexity $\Theta(n\sqrt{n})$ in the worst case.*

Although we could not use the combinatorial analysis of Mitchell *et al.* [9] to obtain better complexity bounds, it is easy to verify that the total number of points for which their algorithm computes the additively weighted Voronoi diagram is $O(n\sqrt{n})$ on a realistic terrain, and therefore the shortest path map for a point can be computed in $O(n\sqrt{n}\log n)$ time.

We are also interested in the maximum complexity of the Voronoi diagram of a set $S$ of $m$ sites on a realistic terrain $\mathcal{T}$, where distances are shortest path distances on $\mathcal{T}$. The Voronoi cell of a site $s_i \in S$ on $\mathcal{T}$ is connected, but not necessarily simply-connected. As a consequence, only $O(m)$ bisectors appear in the Voronoi diagram of $S$ on $\mathcal{T}$, and there are $O(m)$ Voronoi vertices. This immediately leads to an $O(mn\sqrt{n})$ bound on the complexity, but in [11], we show that it is $O(n\sqrt{n})$.

Since the bisector of two sites can have $\Omega(n)$ breakpoints, this is a trivial lower bound. Alternatively, we can place $m$ sites on a terrain such that projected, all $m-1$ bisectors are lines parallel to the $y$-axis. Each bisector intersects $\Omega(\sqrt{n})$ triangles between the boundaries with the terrain, which gives a Voronoi diagram of complexity $\Omega(m\sqrt{n})$.

**Theorem 13** *The Voronoi diagram of a set of $m$ sites on a realistic terrain has complexity $\Omega(n+m\sqrt{n})$ and $O(n\sqrt{n})$ in the worst case.*

## 5 Concluding Remarks

This paper studied realistic input models for polyhedral terrains, a topic that has not been considered so far. We have made three input assumptions that together are necessary and sufficient to show a subquadratic upper bound on the complexity of the visibility map. For paths, bisectors, shortest path maps, and Voronoi diagrams on terrains, we used a fourth input assumption and proved upper and lower bounds on their complexities. Our research helps to explain the discrepancy between the worst-case performance of algorithms on polyhedral terrains and their efficiency in practice.

The upper and lower bounds for visibility maps, shortest paths, and the shortest path map are tight or nearly tight, but there is a considerable gap for bisectors and Voronoi diagrams. This is the most important open problem that arises from this paper.

## References

[1] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. *Algorithmica*, 8:391–406, 1992.

[2] B. Aronov, A. Efrat, V. Koltun, and M. Sharir. On the union of $\kappa$-round objects in three and four dimensions. In *Proc. 20st Annu. ACM Sympos. Comput. Geom.*, pages 383–390, 2004.

[3] J. Chen and Y. Han. Shortest paths on a polyhedron. *Internat. J. Comput. Geom. Appl.*, 6(2):127–144, 1996.

[4] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34:81–97, 2002.

[5] A. Efrat. The complexity of the union of $(\alpha,\beta)$-covered objects. *SIAM J. Comput.*, 34:775–787, 2005.

[6] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proc. 31st Annu. ACM Sympos. Theory of Computing*, pages 770–779, 1999.

[7] M. J. Katz, M. H. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom. Theory Appl.*, 2:223–234, 1992.

[8] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM J. Comput.*, 23:154–169, 1994.

[9] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.

[10] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. *Internat. J. Comput. Geom. Appl.*, 7(4):317–347, Aug. 1997.

[11] E. Moet, M. van Kreveld, and A. F. van der Stappen. On realistic terrains. To appear in *Proc. 22nd Annu. ACM Sympos. Comput. Geom.*, 2006.

[12] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *J. Algorithms*, 21:629–656, 1996.

[13] A. F. van der Stappen, M. H. Overmars, M. de Berg, and J. Vleugels. Motion planning in environments with low obstacle density. *Discrete Comput. Geom.*, 20(4):561–587, 1998.

[14] M. van Kreveld. On fat partitioning, fat covering, and the union size of polygons. *Comput. Geom. Theory Appl.*, 9(4):197–210, 1998.

# River networks and watershed maps of triangulated terrains revisited[*]

Hee-Kap Ahn[†] Mark de Berg[‡] Otfried Cheong[§] Herman Haverkort[¶] A. Frank van der Stappen[||]
Laura Toma[**]

## Abstract

Triangulated surfaces are often used to represent terrains in geographic information systems. We investigate the complexity of river networks and watershed maps on such terrains under the assumption that water always follows the path of steepest descent. We show that the worst-case complexity is only $\Theta(n^2)$ if all triangles are non-obtuse or if all triangles are fat, that is, their minimum angles are bounded from below by a positive constant. Furthermore, we can compute the river networks and watershed maps by tracing paths in a directed acyclic graph representation of the triangulation—a property that can be exploited to do computations I/O-efficiently.

## 1 Introduction

Hydrologists, country planners etc. use terrain models while managing or monitoring water resources, possible flood areas, erosion and other natural processes. In such applications it is important that one can identify where rivers are, where the boundaries of their watersheds are (the areas that drain through them), and to which rivers they are tributaries. Construction works and natural processes in the terrain may change its shape and affect the river network and the boundaries of watersheds. To analyse or predict such changes by manual quantification of watersheds would be a tedious and time-consuming job, so we would rather compute river networks and watershed maps from the updated terrain model automatically.

Therefore several computational geometers have studied the structure, complexity and computation of river networks and watershed maps on triangulated terrains [3, 5, 7, 8]. To be able to discuss their and our results, we first give some definitions. A *terrain* is the graph in $\mathbb{R}^3$ of a continuous function $z = f(x, y)$ defined on a compact, connected subset of the $xy$-plane. A triangulated terrain or TIN is a terrain that consists of triangles. We assume that water always runs downhill in the direction of steepest descent (to keep the definitions simple we assume that the direction of steepest descent is always unique). The watershed of a point $p$ is the set of all points in the terrain from which the water flows to $p$. The drainage area of $p$ is the size of its watershed. The *river network* is the set of points with drainage area greater than zero, or in other words, the set of points whose watersheds are two-dimensional regions. A *watershed map* is a map of the boundaries between the watersheds of points of interest (such as river mouths and confluences).

Yu et al. distinguish three types of edges in the triangulation [8]: *confluent* edges or *channels* are edges that receive water from both adjacent triangles (because on both adjacent triangles, the direction of steepest descent is directed towards the edge); *transfluent* edges receive water from one adjacent triangle, which continues its way down the other triangle; and *diffluent* edges or *ridges* receive no water (because on both adjacent triangles, the direction of steepest descent is directed away from the edge).

De Berg et al. [3] observe that the river network of a triangulated terrain consists of confluent edges and paths of steepest descent that start from the lower ends of confluent edges. McAllister and Snoeyink [5, 7] analyse the complete topology of watershed maps. They find that watersheds are bounded by ridges and by paths of steepest descent that lead to the points for which we want to know the watershed boundaries and to certain vertices in the triangulation. Thus, in a triangulation with $n$ vertices, both the river network and the watershed map consist of edges and vertices already present in the triangulation, plus $O(n)$ paths of steepest descent.

The map can therefore be computed by an algorithm based on tracing paths of steepest descent in the terrain. We can model this as a computation on a directed planar graph: let the *descent graph* $\mathcal{G}$ be the graph that contains a node $v(e)$ for every edge $e$ of the triangulation, and a directed arc from $v(e)$ to $v(f)$ if and only if $e$ and $f$ are on the boundary of the

[*]We thank Sang-Won Bae, Hyo-Sil Kim, and the participants of the 2005 Carleton-Eindhoven workshop on computational geometry for discussions and the stimulating environment in which this research took place.

[†]Department of Computer Science, Korea Advanced Institute of Science and Technology, heekap@gmail.com

[‡]Department of Mathematics and Computer Science, Eindhoven University of Technology, mdberg@win.tue.nl

[§]Department of Computer Science, Korea Advanced Institute of Science and Technology, otfried@kaist.ac.kr. Otfried Cheong was supported by LG Electronics.

[¶]Department of Mathematics and Computer Science, Eindhoven University of Technology, cs.herman@haverkort.net

[||]Department of Information and Computing Sciences, Utrecht University, frankst@cs.uu.nl

[**]Department of Computer Science, Bowdoin College, ltoma@bowdoin.edu

same triangle and there is a path of steepest descent across that triangle from $e$ to $f$. Following a path of steepest descent (until it reaches a vertex or a confluent edge) now corresponds to following a path in $\mathcal{G}$. Computing a watershed map by tracing paths in this graph and connecting them properly can be done in $O(n \log n + k)$ time, where $k$ is the complexity of the output (McAllister [6]).

Alas, the output may be quite big, at least in theory. De Berg et al. [3] describe how to construct a terrain with $n$ vertices that has a descent graph with cycles, so that a path of steepest descent may return to the same edge of the triangulation several times. In fact their terrain contains $\Theta(n)$ rivers that each cross a set of $\Theta(n)$ edges $\Theta(n)$ times. Thus the river network has $\Theta(n^3)$ vertices and so has the watershed map. De Berg et al. provide some Dutch comfort by proving that the worst-case complexity of river networks (and, by the same arguments, watershed maps) is in fact not worse than $\Theta(n^3)$.

*Problem.* When computing watershed maps for large terrains that may not even fit in main memory, a cubic complexity of the watershed map would be disastrous.

On top of that, when the terrain does not fit in main memory, the computation may be slow. Accessing data stored on disk takes much longer than accessing data in main memory: in present-day hardware, the difference may be a factor 1 000 000. This is due to the latency of hard disks. Fortunately, once the disk and its read/write head have been moved into the correct position, reading a large block of data is almost as fast as reading just a few bytes. Hence, to amortize the latency, current computer systems transfer data between disk and memory in blocks: when we access the disk to read an edge of the triangulation, we do not read only one edge, but a block of several thousands.

To run McAllister's algorithm and trace paths of steepest descent without accessing the disk too often, we would have to group the nodes of the descent graph in blocks such that while following paths in this graph, we do not cross block boundaries too often. However, the best known methods for blocking planar graphs perform poorly [1]. Research into I/O-efficient algorithms (algorithms with optimized disk access patterns) has yielded another, much more efficient technique, called time-forward processing [2, 4], that can be used to process directed *acyclic* graphs efficiently. Unfortunately, as is apparent from the aforementioned construction by De Berg et al. [3], the descent graph $\mathcal{G}$ is not guaranteed to be acyclic.

However, the construction by De Berg et al. is highly contrived and we do not expect to encounter such artefacts in practice. This has led us to investigate what easily verifiable, realistic properties of a triangulation would guarantee that $\mathcal{G}$ is acyclic. Thus we kill two birds with one stone: find out under what conditions time-forward processing can be applied, and prove that the complexity of river networks and watershed maps in realistic triangulations is only $O(n^2)$.

*Our results.* We prove that the descent graph is acyclic if all triangles in the triangulation are non-obtuse (either in space, or in the projection on a horizontal plane).

Furthermore, we describe a method to cut the edges of *any* triangulation into segments such that the descent graph on those segments is acyclic. If all triangles in the triangulation are fat—that is, if their minimum angles, either in space or in the projection, are bounded from below by a positive constant—then the number of segments per edge is $O(1)$, and thus the total complexity of the descent graph is $O(n)$.

Therefore, for triangulations with only non-obtuse triangles or only fat triangles the complexity of any steepest descent path is $O(n)$, and the total complexity of river networks and watershed maps is therefore $O(n^2)$. We prove that this is optimal in the worst case by means of a lower-bound construction on a triangulated regular square grid.

Below we describe our results for descent graphs of fat triangulations and our lower-bound construction. We leave the results on non-obtuse triangulations for the full version of this paper.

## 2 Fat triangulations

We first describe our method to cut the edges of any triangulation into segments. For a vertex $v$ in the triangulation, let $r_{\min}(v)$ be the distance to its nearest neighbour vertex in the triangulation, and let $d_{\min}(v)$ be the distance between $v$ and the nearest edge in the triangulation that is not incident to $v$. We define the *neighbourhood* of a vertex $v$ as the disk centered on $v$ with radius $\min\{r_{\min}(v)/2, d_{\min}(v)\}$. The neighbourhood boundaries divide every edge in the triangulation into two *inner segments* (the segments inside the neighbourhoods of the endpoints) and at most one *outer segment* (the rest of the edge)—see Fig. 1. The neighbourhoods are pairwise disjoint.

We further subdivide the outer segments of the edges into a minimum number of smaller segments, called *free segments*, such that each segment's minimum circumscribed circle does not intersect any other edges—see Fig. 1 for an example. All segments are considered to include their upper endpoints and to be relatively open at their lower endpoints.

The *descent graph* $\mathcal{G}$ on these segments contains a node for every inner or free segment; the segment corresponding to node $v$ is denoted by $segm(v)$, and its upper and lower endpoint are indicated by $up(v)$ and $lw(v)$, respectively. The descent graph contains a directed arc from node $a$ to node $b$ if and only if $segm(a)$ and $segm(b)$ are on the boundary of the same triangle $T$ and there is a path of steepest descent across $T$ from $segm(a)$ to $segm(b)$.

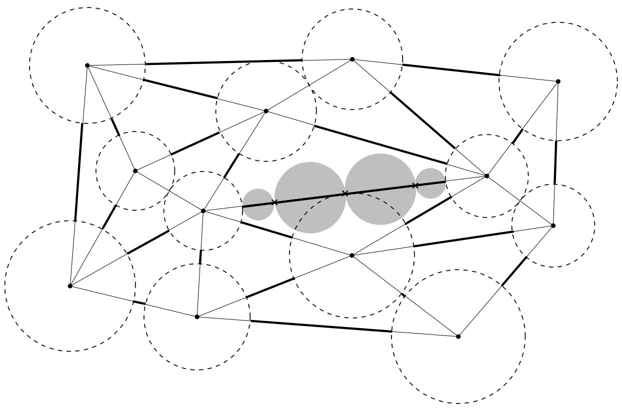


Figure 1: A part of a triangulation. Dotted circles delimit the vertex neighbourhoods. Inner edge segments are drawn with thin lines, outer edge segments are drawn with thick lines. For one outer edge, a subdivision into free segments is shown, with the minimum circumscribed circles of the segments (shaded disks).

For two nodes $a$ and $b$ in $\mathcal{G}$, we define $segm(a) \succ segm(b)$ if and only if $z(up(a)) > z(up(b))$, or $z(up(a)) = z(up(b))$ and $z(lw(a)) > z(lw(b))$, where $z(p)$ is the elevation of $p$.

**Lemma 1** *If $\mathcal{G}$ contains an arc from $a$ to $b$, then $segm(a) \succ segm(b)$.*

**Proof.** Without loss of generality, assume $segm(b)$ is oriented from east to west and $segm(a)$ lies on the triangle $T$ to the north of $segm(b)$. Let $L$(owland) be the closed halfplane bounded from above by the contour line of $T$ through $up(b)$. Let $H$(illside) be the open halfplane that contains $segm(b)$ and is bounded by the line of steepest descent through $lw(b)$ on $T$. Let $N$(orth) be the open halfplane bounded from below by the line that contains $segm(b)$. (See Fig. 2)

Assume, for the sake of contradiction, $segm(a) \not\succ segm(b)$. Then $segm(a)$ lies completely in $L$. Because there is flow across $T$ from $segm(a)$ to $segm(b)$, there is at least one point in $segm(a)$ that lies in $H \cap N$, and hence, in $L \cap H \cap N$. By Thales' Theorem, that point lies inside the minimum circumscribed circle of $segm(b)$.

By definition, the minimum circumscribed circle of any free segment $segm(b)$ cannot intersect any other segments (or vertices) of the triangulation, and the minimum circumscribed circle of an inner segment can only intersect other inner segments in the same vertex neighbourhood. So $segm(a)$ and $segm(b)$ must be inner segments in the neighbourhood of some vertex $v$. By the assumption $segm(a) \not\succ segm(b)$, the endpoint of $segm(b)$ that is not $v$ must lie at least as high as the endpoint of $segm(a)$ that is not $v$. Since $segm(a)$ and $segm(b)$ have the same length, this implies that $segm(b)$ does not descend more steeply (or ascend less steeply) from $v$ than $segm(a)$. Thus the triangle shared by $segm(a)$ and $segm(b)$ is not tilted towards $segm(b)$, which contradicts that there would be a path of steepest descent from $segm(a)$ to $segm(b)$.

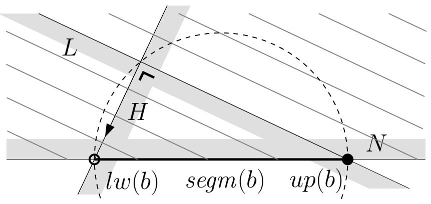


Figure 2: $L \cap H \cap N$ lies in the minimum circumscribed circle of $segm(b)$.

Therefore $segm(a) \succ segm(b)$. ☐

Let an *α-fat triangulation* be a triangulation that only contains triangles that have minimum angle at least $\alpha$ in the projection on a horizontal plane.

**Theorem 2** *The descent graph $\mathcal{G}$ on the segments of an α-fat triangulation with $n$ triangles is a planar directed acyclic graph with $O(n/\alpha^2)$ nodes.*

**Proof.** Sorting the nodes of $\mathcal{G}$ into $\succ$-order puts them into topological order (by Lemma 1); hence $\mathcal{G}$ is acyclic. To bound the number of nodes we use the fact that each edge of the triangulation is cut into at most $2\lceil 2.64/\alpha^2 \rceil$ segments. We omit the details from this abstract. ☐

**Corollary 3** *Any path of steepest descent or ascent in an α-fat triangulation has $O(n/\alpha^2)$ vertices, and the total complexity of the river network and the watershed map in an α-fat triangulation is $O(n^2/\alpha^2)$.*

To allow tracing paths of steepest descent through channels and vertices, $\mathcal{G}$ also needs to include nodes representing the vertices of the triangulation. With a more refined definition of vertex neighbourhoods the theorem also holds if angles of triangles are measured in space instead of in the projection on the plane.

## 3 Lower bound

We describe a terrain, represented by a regular grid of $n/2$ triangulated squares, that has a river network with $\Theta(n^2)$ vertices. The basic ingredient of our construction is shown in Fig. 3. It is a terrain of $5 \times 7$ squares. The boundary is at roughly the same elevation all around, except for three lower, horizontal edges $a$, $b$ and $z$ (and the surrounding slopes that connect them to the higher boundary edges). All rivers that enter the terrain across the interior of $a$ or $b$ leave the terrain as separate rivers across the interior of $z$. The shaded areas drain through two more rivers that leave the terrain across the interior of $z$. All of the above properties can easily be maintained if the valleys (the triangles across which water flows from $a$ or $b$ to $z$) are simultaneously raised or lowered.

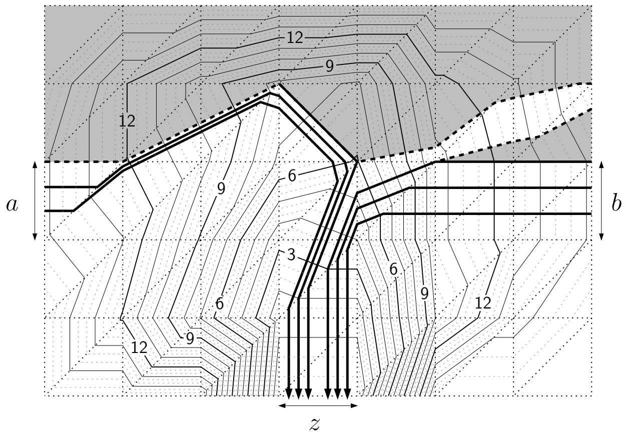


Figure 3: A contour map of a terrain of $5 \times 7$ triangulated squares. The rivers that enter the terrain across $a$ or $b$ leave the terrain as separate rivers across $z$.

**Lemma 4** *For any $k > 0$ there is a rectangular terrain $T(k)$ of less than $168 \times 2^k$ triangles such that at least $2^k$ different rivers flow out of the terrain across the middle edge of one of the long sides.*

**Proof.** Let $T(1)$ be the terrain shown in Fig. 3, with the valleys raised so that $a$ and $b$ are at the same elevation as the rest of boundary (except $z$). For $k > 1$, the terrain $T(k)$ consists of two appropriately rotated and mirrored copies of $T(k-1)$, connected by a copy of $T(1)$ with valleys appropriately lowered—see Fig. 4. We complete the terrain to a rectangle by padding it with triangles, such that the rivers that flow across $z$ flow straight to the closest edge of the boundary. The claimed bounds are easily proven by induction. □

**Theorem 5** *There is a terrain of $n$ non-obtuse, fat, Delaunay triangles that has a river network with $\Theta(n^2)$ vertices.*

**Proof.** Take $T(\lfloor \log(n/336) \rfloor)$ and complete it to $n$ triangles, such that all $\Omega(n)$ rivers that flow from it cross $\Theta(n)$ edges of the complementary triangles. □

## 4 Discussion

De Berg et al. [3] asked if one could prove an $O(n^2)$ bound on the complexity of river networks in Delaunay triangulations or other triangulations with well-shaped triangles. We showed that this bound indeed holds if all triangles are non-obtuse, or if the triangles are fat. The question if the same bound can be proven for a Delaunay triangulation is still open.

In the case of fat triangles, the constant in the $O(n^2)$ bound depends on the minimum angle of the triangles. A close look at the analysis reveals that the dependency seems to be quite local: only in the direct neighbourhood of small angles, will many nodes be put in the descent graph. Hence an occasional non-fat

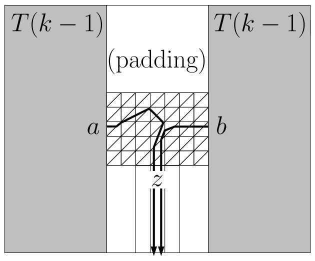


Figure 4: $T(k)$: a terrain of $O(2^k)$ triangles with $\Theta(2^k)$ rivers flowing out across a single edge.

triangle in the triangulation will not affect the bounds on the complexity of the river network significantly.

Our lower bound proves that the $O(n^2)$ worst-case bound cannot be improved under any of the conditions mentioned so far. However, because quadratic complexity would still be impractical, because our lower-bound construction is still quite contrived, and because Yu et al. [8] observed linear complexity in experiments on real data, we still wonder if other, easily verifiable conditions on triangulations may enable us to prove subquadratic bounds.

## References

[1] P. K. Agarwal, L. Arge, T. M. Murali, K. R. Varadarajan and J. S. Vitter. I/O-efficient algorithms for contour-line extraction and planar graph blocking. *Symp. on Discrete Algorithms '98*, p117–126.

[2] L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.

[3] M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink and S. Yu. The complexity of rivers in triangulated terrains. *Canad. Conf. Comp. Geom. '96*, p325–330.

[4] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff and J. S. Vitter. External-memory graph algorithms. *Symp. on Discrete Algorithms '95*, p139–149.

[5] M. McAllister. *The computational geometry of hydrology data in geographic information systems.* PhD th., Univ. of British Columbia, 1999.

[6] M. McAllister. *A watershed algorithm for triangulated terrains. Canad. Conf. Comp. Geom. '99.*

[7] M. McAllister and J. Snoeyink. Extracting consistent watersheds from digital river and elevation data. *Ann. Conf. Amer. Soc. for Photogrammetry and Remote Sensing / Amer. Congr. on Surveying and Mapping '99.*

[8] S. Yu, M. van Kreveld and J. Snoeyink. Drainage Queries in TINs: from local to global and back again. *Symp. on Spatial Data Handling '96*, p13A.1–14.

# In-Place Randomized Slope Selection

Henrik Blunck* Jan Vahrenhold†

## Abstract

*Slope selection*, i.e. selecting the slope with rank $k$ among all $\binom{n}{2}$ lines induced by a collection $\mathcal{P}$ of points, results in a widely used robust estimator for line-fitting. In this paper, we demonstrate that it is possible to perform slope selection in expected $\mathcal{O}(n \cdot \log_2 n)$ time using only constant extra space in addition to the space needed for representing the input.

## 1 Introduction

Computing a *line estimator*, i.e., fitting a line to a collection $\mathcal{P}$ of $n$ data points $\{p_1, \ldots, p_n\}$ in the plane is a frequent task in statistical analysis. A frequently used robust line estimator is the so-called *Theil-Sen* estimator (see [13] and the references therein) which considers all $\binom{n}{2}$ lines induced by the points in $\mathcal{P}$ and selects the line with median slope. This problem is also known as the (median) *slope selection* problem and has been shown to exhibit an $\Omega(n \cdot \log_2 n)$ lower bound [6]. Several deterministic algorithms for solving this problem in optimal $\mathcal{O}(n \cdot \log_2 n)$ running time have been presented [5, 6, 10], however, as noted by Matoušek *et al.* [13], they are based on relatively complicated concepts such as parametric search, sorting network, expander graphs, or cuttings. More practical approaches have resulted in randomized algorithms with expected $\mathcal{O}(n \cdot \log_2 n)$ running time [7, 12, 15].

**The Model** The goal of investigating space-efficient algorithms is to design algorithms that use very little extra space in addition to the space used for representing the input. The input is assumed to be stored in an array `A` of size $n$, thereby allowing random access. We assume that a constant size memory can hold a constant number of words. Each word can hold one pointer, or an $\mathcal{O}(\log_2 n)$ bit integer, and a constant number of words can hold one element of the input array. An *in-place* algorithms uses $\mathcal{O}(1)$ extra words of memory. Recently, a number of in-place algorithms have been designed for solving geometric problems—see, e.g., [1, 2, 3, 4, 16].

In addition to theoretical considerations, one reason for investigating space-efficient algorithms is that they have the potential of using the different stages of hierarchical memory, e.g., caches, to a much higher degree of efficiency. Another motivation, especially for designing algorithms for statistical data analysis, comes from the recently increased interest in sensor networks where small-scale computing devices are used to collect large amounts of data. Since the memory of such sensor devices usually is very limited, and since transmitting data is much more costly than local computation, it is desirable to process as much data as possible locally before transmitting (intermediate) results.

**Our Results** In this paper we show how to solve the slope selection problem in expected optimal $\mathcal{O}(n \log_2 n)$ time while at the same time using only constant extra space. Our algorithm follows the approach of Matoušek [12], and, in the course of implementing his algorithm in-place, we also devise an in-place variant of the so-called *randomized interpolation search* technique. This variant, together with an algorithmic subroutine for efficiently constructing and storing a set of randomly sampled intersections, is of independent interest, since it can be used as a substitute for Megiddo's *parametric search* technique [14].

## 2 Randomized Interpolation Search

In the following, the slope selection problem is studied in the dual setting where each point $(x, y)$ is identified with the line $\{(\xi, \upsilon) \mid \upsilon = x \cdot \xi - y\}$ and vice versa. Selecting the $k$-th smallest slope is dual to the following problem: Given a set of $n$ lines in the plane, find the $k$-th leftmost intersection point induced by the arrangement of the lines. Since the duality transform can be performed in an implicit way, we will assume that our input is given as a set $\mathcal{P}$ of lines in the plane.

Since it is infeasible to compute all $\Theta(n^2)$ intersections induced by $\mathcal{P}$, the algorithm of Matoušek [12] maintains a vertical strip $\langle b, e\rangle := [b, e] \times \mathbb{R} \subset \mathbb{R}^2$ that is guaranteed to contain the $k$-th smallest intersection point. For a parameter $r$ (to be defined later), the algorithm first constructs a sample $\mathcal{R}$ of size $r$ drawn from the intersections inside $\langle b, e\rangle$. It then selects two intersections from $\mathcal{R}$ whose $x$-coordinates are used to construct a (narrower) candidate strip $\langle b', e'\rangle$. The algorithm then checks whether $\langle b', e'\rangle$ indeed contains the $k$-th smallest intersection point. If this is not the

*Westfälische Wilhelms-Universität Münster, Institut für Informatik, Einsteinstr. 62, 48149 Münster, Germany. E-mail: `blunck@math.uni-muenster.de`

†Westfälische Wilhelms-Universität Münster, Institut für Informatik, Einsteinstr. 62, 48149 Münster, Germany. E-mail: `jan@math.uni-muenster.de`

case, the process is repeated for $\langle b, e\rangle$ but using a new sample $\mathcal{R}$, otherwise the algorithm iterates with the refined strip $\langle b', e'\rangle$. The iteration terminates when the number $|\mathcal{I}(b, e)|$ of intersections within $\langle b, e\rangle$ is no larger than $r$: in this case, the $k$-th leftmost intersection point can be computed directly by enumerating all intersections in $\langle b, e\rangle$ and selecting the appropriate one. This refinement strategy is referred to as *randomized interpolation search*—see [12]. The efficiency of the resulting algorithm for slope selection is based upon the following lemma which (applied iteratively) implies that the number $|\mathcal{I}(b, e)|$ of intersections that lie inside $\langle b, e\rangle$ can be reduced to $\mathcal{O}(r)$ using an expected constant number of iterations:

**Lemma 1 (Lemma 2.1 in [13])** *Given a set of numbers $\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$, an index $k$ $(1 \leq k \leq N)$, and an integer $r > 0$, we can compute in $\mathcal{O}(r)$ time an interval $[\theta_{lo}, \theta_{hi}]$, such that, with probability $1-1/\Omega(\sqrt{r})$, the $k$-th smallest element of $\Theta$ lies within this interval, and the number of elements in $\Theta$ that lie within the interval is at most $N/\Omega(\sqrt{r})$.*

The above lemma will be applied for $N \in \mathcal{O}(n^2)$. Furthermore, Matoušek *et al.* [13] proved that we may choose $r := \lceil n^{\beta}\rceil$ for any $0 < \beta < 1$ without affecting the asymptotic efficiency of the resulting algorithm, and thus we will set $r := \lceil\sqrt{n}\rceil$. As we will see below, our in-place algorithm will be working with binary encoded numbers, and thus accessing a single number $\theta_i$ will take $\mathcal{O}(\log_2 n)$ time. Thus, the in-place version of the algorithm implied by Lemma 1 runs in $\mathcal{O}(r \cdot \log_2 n)$ time.

It remains to describe how to construct (and store!) the sampled set $\mathcal{R}$ of $r = \lceil\sqrt{n}\rceil$ intersections in an in-place setting, i.e., using only constant extra space. Furthermore, we need to discuss how to compute $|\mathcal{I}(b, e)|$. To this effect, we describe an algorithm for the first task, which also provides a solution for the second one. Anticipating the results presented in the next section, we combine them with the above lemma and the original analyses of Matoušek *et al.* [12, 13]:

**Theorem 2** *The slope selection problem for a set of $n$ input points in the plane can be solved in-place and in expected optimal $\mathcal{O}(n \cdot \log_2 n)$ running time.*

## 3 Constructing the Random Sample $\mathcal{R}$

Following the approach of Matoušek [12, Lemma 1], we first draw (with replacement) a set of $r$ random integers from $\{0, \ldots, |\mathcal{I}(b, e)| - 1\}$ where $|\mathcal{I}(b, e)|$ is the number of intersections in $\langle b, e\rangle$; these numbers give the ranks of the intersections that will be part of $\mathcal{R}$ with respect to the order in which they are found.

The main ingredient used for efficiently processing intersections in $\langle b, e\rangle$ is the following well-known observation: the number of intersections inside $\langle e, b\rangle$ is exactly the number of inversions between the permutation of $\mathcal{P}$ that arranges the lines in sorted $<_b$-order (the vertical order at $x = b$) and the permutation that arranges the lines in sorted $<_e$-order (the vertical order at $x = b$). Thus, to efficiently compute $|\mathcal{I}(b, e)|$, we can run the classic divide-and-conquer algorithm for inversion counting—see, e.g., [11]. While doing so, we keep track of the total number of inversions/intersections seen so far and "record" an intersection if its rank matches one of the $r$ given ranks. If the ranks are sorted, we can process them in constant extra time per inversion counting operation. We process the "recursion tree" of (our adaption of) the inversion counting algorithm (see Algorithm 1) in a bottom-up, level-by-level traversal, i.e., without the need of maintaining a recursion stack. Since we need to maintain the $r$ ranks and the intersections computed so far in an in-place setting, the algorithm is divided into three phases: During the first phase, we process the lines stored in $\mathtt{A}[0, \ldots, n/2 - 1]$ and use $\mathtt{A}[n/2, \ldots, n - 1]$ to encode the ranks and the intersections found so far. We then reverse the roles of both subarrays, and finalize the algorithm with a third phase that processes the intersections induced by lines stored in different halves of the array.

**Three In-Place Data Structures** For maintaining more than $\mathcal{O}(1)$ numbers or indices, we resort to a standard technique in the design of space-efficient algorithms, namely to encode a single bit by a permutation of two objects $q$ and $r$: For lines $q, r$ with $q <_b r$, the permutation $qr$ encodes a binary zero, and the permutation $rq$ encodes a binary one. We use the subarray of size $n/2$ that does not contain the lines to be processed in the current phase to represent three (implicit) data structures $\mathcal{D}_R$, $\mathcal{D}_L$, and $\mathcal{D}_I$ that occupy a subarray of size $4 \cdot r \cdot \log_2 n$ each:

| | | | | |
|---|---|---|---|---|
| Lines to be processed | | $\mathcal{D}_R$ | $\mathcal{D}_L$ | $\mathcal{D}_I$ |

$0$ $\qquad\qquad\qquad \frac{1}{2}n \qquad\qquad\qquad n-1$

**Storing Ranks** The randomly generated ranks in the range $[0, \ldots, n^2-1]$ are encoded in a "sorted-list" data structure $\mathcal{D}_R$. At initialization of $\mathcal{D}_R$, these ranks are sorted using *heapsort* [17], which performs $\mathcal{O}(r \cdot \log_2 r)$ operations each of which requires decoding a binary-encoded integer or swapping the values of two "rank elements".[1] This results in $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$ time spent for sorting. Having sorted the ranks, our algorithm will be able to traverse the list and report each rank to be processed in $\mathcal{O}(\log_2 n)$ time.

[1]Note, that swapping two encoded values $a$, $b$ (as done by heapsort) does not require swapping the *blocks* of input elements used for encoding $a$ and $b$—it merely involves updating the *permutations* used to represent bits. Therefore, each input element will be at most one position off its correct position.

**Storing Lines Involved in Intersections** We use a "sorted-list" data structure $\mathcal{D}_L$ to record (references to) lines involved in all of the sampled intersections found so far. These (references to) lines are maintained in sorted $<_b$-order. Every reference to a line is inserted into $\mathcal{D}_L$ using insertion sort (ignoring duplicates), and this leads to $\mathcal{O}(r^2 \cdot \log_2 n)$ global cost for maintaining $\mathcal{D}_L$.

**Storing Intersections** The "linked-list" data structure $\mathcal{D}_I$ records the intersections found so far by indexing into $\mathcal{D}_L$. To add an intersection induced by two lines $\ell_1$ and $\ell_2$ to $\mathcal{D}_I$, we first insert references to $\ell_1$ and $\ell_2$ in sorted $<_b$-order into $\mathcal{D}_L$ and then append the pair $(i, j)$ referencing the references in $\mathcal{D}_L$ to these two lines at the end of $\mathcal{D}_I$. The cost for performing a single insert to $\mathcal{D}_I$ is $\mathcal{O}(\log_2 n)$, and thus we have a global update cost of $\mathcal{O}(r \cdot \log_2 n)$.

## 3.1 Processing one Half of the Subarray

The algorithms for processing the two halves of $\mathtt{A}[0, \ldots, n-1]$ are symmetric, and thus we present the algorithm for processing the subarray $\mathtt{A}[0, \ldots, n/2-1]$.

**Counting Inversions** The algorithm for counting *all* inversions in $\langle b, e\rangle$ is an extension of the iterative *mergesort* algorithm: starting from the set of lines in sorted $<_b$-order, the algorithm iteratively merges the lines into $<_e$-order while counting inversions. During each *merge*-step of the algorithm, two subarrays already in sorted $<_e$-order are merged into a single $<_e$-sorted subarray. Each of these subarrays has been processed during the previous iteration, and thus all inversions involving lines from only one of these subarrays have been processed. An obvious, yet crucial, fact guaranteeing the correctness of the inversion counting algorithm is that any two subarrays $\mathtt{A}_1$ and $\mathtt{A}_2$ that are merged in the $j$-th iteration of processing the $m$-th bottom-most level of the recursion tree are of the form $\mathtt{A}_1 := \mathtt{A}[j \cdot 2^m, \ldots, (j+1) \cdot 2^m - 1]$ and $\mathtt{A}_2 := \mathtt{A}[(j+1) \cdot 2^m, \ldots, (j+2) \cdot 2^m - 1]$[2]. Since in our case all lines are initially sorted in $<_b$-order, for every invocation of the algorithm CountAndRecord (Algorithm 1) depicted below the following holds: Each line in $\mathtt{A}_1$ precedes all lines in $\mathtt{A}_2$ with respect to the $<_b$-order and the union $\mathtt{A}_1 \cup \mathtt{A}_2$ forms a complete interval of the input in $<_b$-order.

Algorithm 1 can be implemented using constant extra space, and, excluding the time needed for recording the relevant intersections, its running time is linear in the size of the union of the two subarrays to be merged. Thus, excluding the time needed for updating $\mathcal{D}_R$, $\mathcal{D}_L$, and $\mathcal{D}_I$, its time complexity is in

[2]The algorithm can be easily modified to handle instances where $n$ is not a power of two [1].

**Algorithm 1** CountAndRecord($\mathtt{A}_1, \mathtt{A}_2, \langle b, e\rangle$) increments the global count $c$ of intersections (falling inside $\langle b, e\rangle$) by the number of intersections induced by lines in $\mathtt{A}_1$ and $\mathtt{A}_2$ while recording intersections to be sampled in $\mathcal{D}_I$.

```
Require: $A_1$ and $A_2$ are sorted according to $<_e$.
Ensure: $A_1$ and $A_2$ are sorted according to $<_e$.
 1: Let $i_1 := 0$ and $i_2 := 0$.
 2: for $i = 0$ to length$(A_1 \cup A_2) - 1$ do
 3:    Let $\ell_1 := A_1[i_1]$ and $\ell_2 := A_2[i_2]$.
         {The $i$-th element in sorted order is $\ell_1$ or $\ell_2$.}
 4:    if $\ell_1 <_b \ell_2$ then
 5:       Let $c_{i_1} := i_2$.   {$\sharp$(inversions induced by $\ell_1$)
                        $= \sharp$(elements in $A_2$ preceding $\ell_1$)}
 6:       for each rank $\rho$ in $\mathcal{D}_R \cap [c, \ldots, c + c_{i_1}]$ do
 7:          Let $\ell$ be the line stored at $A_2[\rho - c]$.
 8:          Update $\mathcal{D}_I$ to record the pair $(\ell_1, \ell)$ as the
             intersection with rank $\rho$.
 9:       end for
10:       Let $c := c + c_{i_1}$.          {Count intersections.}
11:       $i_1 := i_1 + 1$.                {Advance $i_1$.}
12:    else
13:       $i_2 := i_2 + 1$.                {Advance $i_2$.}
14:    end if
15: end for
```

$\mathcal{O}(n \log_2 n)$. Also, leaving out the code in Lines 6–9, we may use Algorithm 1 to compute $|\mathcal{I}(b, e)|$.

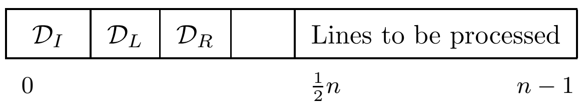
**Merging Two Subarrays Into Sorted $<_e$-Order** It remains to discuss the actual merging process that is required to merge $\mathtt{A}_1$ and $\mathtt{A}_2$ into sorted $<_e$-order. Since each of the subarrays is sorted according to $<_e$, a simple application of the linear-time merging algorithm of Geffert *et al.* [8] will produce the desired result. However, we also need to update the values stored in $\mathcal{D}_L$, since they reference the lines involved in the intersections found so far by directly indexing into $\mathtt{A}$. Merging $\mathtt{A}_1$ and $\mathtt{A}_2$ seems to corrupt the information recorded in $\mathcal{D}_L$, but fortunately the information of what goes where can be computed on the fly while running CountAndRecord: During the $i$-th iteration, this algorithm computes the element with rank $i$ in the final sorted order (Line 3 of Algorithm 1), and thus we simply check whether the line $\ell$ that will be the $i$-th element in sorted order is involved in an intersection. In this case, we simply update the reference to $\ell$ to point to $\ell$'s position after the merge step: the $i$-th position in the union of $\mathtt{A}_1$ and $\mathtt{A}_2$.

We point out that we have to be very careful when maintaining (in $\mathcal{D}_I$) the references to elements stored in $\mathcal{D}_L$: with every new intersection recorded, some references may need to change their position in order to maintain the $<_b$-order. Due to space constraints, we omit the description of the update algorithm, which is summarized in the following lemma:

**Lemma 3** *The global extra cost incurred by updating the references stored in the data structure $\mathcal{D}_L$ while merging subarrays is in $\mathcal{O}(r \cdot \log_2 r \cdot \log_2^2 n)$.*

## 3.2 Finishing Up

After we have processed the first half of the input array (using the second half to maintain the data structures $\mathcal{D}_R$, $\mathcal{D}_L$, and $\mathcal{D}_I$), we reverse the roles of the two subarrays. To do so, we first need to copy the contents of the data structures to the first half of the array. The important detail to keep in mind is that, as a result of the inversion-counting algorithm, the first half of the array is sorted according to $<_e$. Thus, when copying the contents of the data structures to the first half of the array, the order according to which we have to decide whether two lines encode a binary zero or a binary one, is the $<_e$-order.



Having run our subroutine on $\mathtt{A}[n/2, \ldots, n-1]$, we need to finalize the algorithm by processing all intersections induced by lines stored in different halves of the array. As it turns out, we do not need to actually *merge* the lines in $\mathtt{A}[0, \ldots, n/2-1]$ and $\mathtt{A}[n/2, \ldots, n-1]$—it is sufficient to count the inversions and construct the needed intersections. This means that we can simply run Algorithm 1 without the modifications needed to record the "what-goes-where" information. Since the binary encoding scheme permutes only adjacent elements, the lines used to encode the data structures $\mathcal{D}_R$, $\mathcal{D}_L$, and $\mathcal{D}_I$ are at most one position off their correct position (in sorted $<_e$-order). Thus, we can process them with constant extra (look-ahead) space.

As a result of this final invocation of CountAndRecord, the data structure $\mathcal{D}_I$ will reference $r$ pairs of entries in $\mathcal{D}_L$ which in turn reference pairs of lines in $\mathtt{A}$. To select the two intersection points whose $x$-coordinates delimit the candidate strip $\langle b', e' \rangle$ that might be used during the next iteration (see [12]), we could run a selection algorithm. However, since we have chosen $r$ small enough, we can simply sort the pairs in $\mathcal{D}_I$ according to the $x$-coordinate of their intersection. The running time for the sorting (including the time for resolving one level of indirection) is $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$.

Combining the above, Lemma 3, and the fact that $r \in \mathcal{O}(\sqrt{n})$, we obtain the following lemma:

**Lemma 4** *A random sample $\mathcal{R}$ of $r = \lceil \sqrt{n} \rceil$ intersections inside a strip $\langle b, e \rangle$ can be constructed in-place and in $\mathcal{O}(n \cdot \log_2 n)$ time.*

The same algorithm can be used to explicitly construct *all* of the at most $r$ intersection points falling inside $\langle b', e' \rangle$ in the final iteration of the slope selection algorithm. This finishes the proof of Theorem 2.

## References

[1] P. Bose, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and J. Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Computational Geometry: Theory & Applications*, 2006. To appear.

[2] H. Brönnimann and T. M.-Y. Chan. Space-efficient algorithms for computing the convex hull of a simple polygonal line in linear time. In *Proc. Latin American Theoretical Informatics*, LNCS 2976, pp. 162–171. 2004.

[3] H. Brönnimann, T. M.-Y. Chan, and E. Y. Chen. Towards in-place geometric algorithms. In *Proc. 20th Symp. Computational Geometry*, pp. 239–246. 2004.

[4] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. T. Toussaint. Space-efficient planar convex hull algorithms. *Theoretical Computer Science*, 321(1):25–40, 2004.

[5] H. Brönnimann and B. M. Chazelle. Optimal slope selection via cuttings. *Computational Geometry: Theory and Applications*, 10(1):23–29, 1998.

[6] R. Cole, J. S. Salowe, W. L. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Computing*, 18(4):792–810, Aug. 1989.

[7] M. B. Dillencourt, D. M. Mount, and N. S. Nethanyahu. A randomized algorithm for slope selection. *Intl. J. Computational Geometry and Applications*, 2(1):1–27, 1992.

[8] V. Geffert, J. Katajainen, and T. Pasanen. Asymptotically efficient in-place merging. *Theoretical Computer Science*, 237(1–2):159–181, 2000.

[9] P. Huber. *Robust Statistics*. Wiley, 1981.

[10] M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Information Processing Letters*, 47(3):115–122, 1993.

[11] J. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2006.

[12] J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters*, 39(4):183–187, 1991.

[13] J. Matoušek, D. M. Mount, and N. S. Nethanyahu. Efficient randomized algorithms for the repeated median line estimator. *Algorithmica*, 20(2):136–150,1998.

[14] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[15] L. Shafer and W. L. Steiger. Randomizing optimal geometric algorithms. In *Proc. 5th Canadian Conf. Computational Geometry*, pp. 133–138, 1993.

[16] J. Vahrenhold. Line-segment intersection made in-place. In *Proc. 9th Intl. Workshop Algorithms and Data Structures*, LNCS 3608, pp. 146–157, 2005.

[17] J. W. J. Williams. Algorithm 232: Heapsort. *Comm. ACM*, 7(6):347–348, 1964.

# In-Place Algorithms for Computing (Layers of) Maxima

Henrik Blunck* Jan Vahrenhold†

**Abstract**

We describe space-efficient algorithms for solving problems related to finding maxima among points in two and three dimensions. Our algorithms run in optimal $\mathcal{O}(n \log_2 n)$ time and require $\mathcal{O}(1)$ space in addition to the representation of the input.

## 1 Introduction

In this paper, we consider the fundamental geometric problems of computing the maxima of point sets in two and three dimensions and of computing the layers of maxima in two dimensions. Given two points $p$ and $q$, the point $p$ is said to *dominate* the point $q$ iff the coordinates of $p$ are larger than the coordinates of $q$ in all dimensions. A point $p$ is said to be a *maximal point* (or: a *maximum*) of $\mathcal{P}$ iff it is not dominated by any other point in $\mathcal{P}$. The union $\texttt{MAX}(\mathcal{P})$ of all points in $\mathcal{P}$ that are maximal is called the *set of maxima* of $\mathcal{P}$. This notion can be extended in a natural way to compute *layers* of maxima [7]: After $\texttt{MAX}(\mathcal{P})$ has been identified, the computation is iterated for $\mathcal{P} := \mathcal{P} \setminus \texttt{MAX}(\mathcal{P})$, i.e., the next layer of maxima is computed until $\mathcal{P}$ becomes empty.

**Related Work** The problem of finding maxima of a set of $n$ points has a variety of applications in statistics, economics, and operations research (as noted by Preparata and Shamos [14]), and thus was among the first problems having been studied in Computational Geometry: In $\mathbb{R}^2$ and $\mathbb{R}^3$, the best known algorithm, Kung, Luccio, and Preparata's algorithm [11], identifies the set of maxima in $\mathcal{O}(n \log_2 n)$ time which is optimal since the problem exhibits a sorting lower bound [11, 14]. For constant dimensionality $d \geq 4$, their divide-and-conquer approach yields an algorithm with $\mathcal{O}(n \log_2^{d-2} n)$ running time, and Matoušek [12] gave an $\mathcal{O}(n^{2.688})$ algorithm for the case $d = n$. The problem has also been studied for dynamically changing point sets in two dimensions [10] and under assumptions about the distribution of the input points in higher dimensions [1, 8]. Buchsbaum and Goodrich [7] presented an $\mathcal{O}(n \log_2 n)$ algorithm for computing the layers of maxima for point sets in three dimensions. Their approach is based on the plane-sweeping paradigm and relies on dynamic fractional cascading to maintain a point-location structure for dynamically changing two-dimensional layers of maxima. The maxima problem has also been actively investigated in the database community following the definition of the SQL "`skyline`" operator [2] that is used to compute the set of maxima. Spatial index structures have been used to produce the "skyline" practically efficient and/or in a progressive way, that is outputting results while the algorithm is running—see [13] and the references therein. For none of these approaches, non-trivial upper bounds are known.

**The Model** The goal of investigating space-efficient algorithms is to design algorithms that use very little extra space in addition to the space used for representing the input. The input is assumed to be stored in an array `A` of size $n$, thereby allowing random access. We assume that a constant size memory can hold a constant number of words. Each word can hold one pointer, or an $\mathcal{O}(\log_2 n)$ bit integer, and a constant number of words can hold one element of the input array. The extra memory used by an algorithm is measured in terms of the number of extra words; an *in-place* algorithm uses $\mathcal{O}(1)$ extra words of memory. It has been shown that some fundamental geometric problems such as 2D convex hulls and closest pairs can be solved *in-place* and in optimal time [3, 4, 6]. More involved problems (range searching, line-segment intersection) can be (currently) solved *in-place* only if one is willing to accept near-optimal running time [5, 15], and 3D convex hulls and related problems seem to require both (poly-)logarithmic extra space and time [5].

**Our Contribution** The main issue in designing in-place algorithms is that most powerful algorithmic tools (unbalanced recursion, sweeping, multi-level data structures, fractional cascading) require $\Omega(\log_2 n)$ or even $\Omega(n)$ extra space, e.g., for the recursion stack or pointers. This raises the question of whether there exists a time-space trade-off for geometric algorithms besides range-searching. We make a further step towards a negative answer to this question by demonstrating that $\mathcal{O}(1)$ extra space is suf-

*Westfälische Wilhelms-Universität Münster, Institut für Informatik, Einsteinstr. 62, 48149 Münster, Germany. E-mail: `blunck@math.uni-muenster.de`

†Westfälische Wilhelms-Universität Münster, Institut für Informatik, Einsteinstr. 62, 48149 Münster, Germany. E-mail: `jan@math.uni-muenster.de`

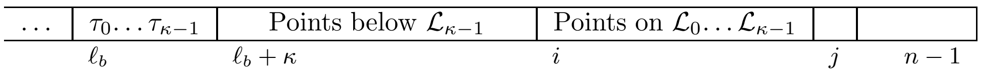

$\dots$
$\tau_0 \dots \tau_{\kappa-1}$
Points below $\mathcal{L}_{\kappa-1}$
Points on $\mathcal{L}_0 \dots \mathcal{L}_{\kappa-1}$
$\ell_b$
$\ell_b + \kappa$
$i$
$j$
$n-1$

Figure 1: Data layout for processing the topmost $\kappa$ layers.

ficient to obtain optimal $\mathcal{O}(n \log_2 n)$ algorithms for computing skylines in two and three dimensions and two-dimensional layers of maxima. The solution to the latter problem is of particular interest since it is the first optimal in-place algorithm for a geometric problem that is not amenable to a solution based on balanced divide-and-conquer or Graham's scan.

## 2 Computing the Skyline in $\mathbb{R}^2$ and $\mathbb{R}^3$

A point $p$ from a point set $\mathcal{P}$ is said to be *maximal* if no other point from $\mathcal{P}$ has larger coordinates in *all* dimensions; ties are broken using a standard shearing technique. This definition has been transferred by Kung *et al.* [11] into a plane-sweeping algorithm for the two-dimensional case and a divide-and-conquer approach for the higher-dimensional case. The output of the algorithm will consist of a permutation of the input array A and an index $k$ such that $k$ points constituting the set of maxima are stored sorted by decreasing $y$-coordinates in $\mathtt{A}[0, \dots, k-1]$.

**Lemma 1** *The* skyline, *i.e., the set of maxima of a set $\mathcal{P}$ of $n$ points in two dimensions can be computed in-place and in optimal time $\mathcal{O}(n \log_2 n)$. If $\mathcal{P}$ is sorted according to $<_y$, the running time is in $\mathcal{O}(n)$.*

For the case of a three-dimensional input, we implement Kung *et al.*'s [11] divide-and-conquer algorithm using an in-place divide-and-conquer scheme we have proposed earlier [3]; this scheme is based on in-place routines for median-finding, partitioning, and merging. Since we cannot explicitly keep track of the number of maxima in each subproblem, we have to recover them algorithmically during each merging step.

**Theorem 2** *The* skyline, *i.e., the set of maxima of an $n$-element point set in three dimensions can be computed in-place and in optimal time $\mathcal{O}(n \log_2 n)$.*

## 3 Computing the Layers of Maxima in $\mathbb{R}^2$

A naïve approach to computing all layers of maxima would be to iteratively use the in-place algorithm described in Section 2. Since a point set may exhibit a linear number of layers, this will lead to $\mathcal{O}(n^2 \log_2 n)$ worst-case running time. In this section, we will show that we can simultaneously peel off multiple layers such that the resulting algorithm runs in optimal $\mathcal{O}(n \log_2 n)$ time; its goal is to rearrange the input such that the points are grouped by layers and each layer is sorted by decreasing $y$-coordinate.

### 3.1 Computing the Topmost $\kappa$ Layers

Our algorithm imitates counting-sort, i.e., prior to actually partitioning the points into layers, it first computes the number of points for each of the layers. This can be done efficiently:

**Lemma 3** *The number of layers of maxima exhibited by an $n$-element point set in two dimensions can be computed in-place and in $\mathcal{O}(n \log_2 n)$ time.*

**Counting the points on the topmost $\kappa$ layers** In this section, we prove the following lemma:

**Lemma 4** *The cardinality $c_i$ of each of the topmost $\kappa$ layers of $\mathtt{A}[0, \dots, n-1]$ can be computed in $\mathcal{O}(n \log_2 n)$ time. If the points are presorted, the complexity is $\mathcal{O}(n + \xi \log_2 \kappa)$ where $\xi = \sum_{i=0}^{\kappa-1} c_i$.*

To illustrate the algorithm, let us assume that we have already peeled off some layers and stored the result in $\mathtt{A}[0, \dots, \ell_b - 1]$. Inductively, we maintain the following invariant which, prior to the first iteration, can be established by sorting and by setting $\ell_b := 0$:

**Invariant (SORT):** The points that have not yet been assigned to a layer are stored in $\mathtt{A}[\ell_b, \dots, n-1]$ and are sorted by decreasing $y$-coordinate.

Let us further assume that the total number of points on the topmost $\kappa$ layers $\mathcal{L}_0$ through $\mathcal{L}_{\kappa-1}$ of the *remaining* points stored in $\mathtt{A}[\ell_b, \dots, n-1]$ is $\xi$ and that $\ell_b + 2\xi \leq n$. The first step then is to stably extract the $\xi$ points on the topmost $\kappa$ layers and move them to $\mathtt{A}[\ell_b, \dots, \ell_b + \xi - 1]$ while maintaining the sorted $y$-order in $\mathtt{A}[\ell_b + \xi, \dots, n-1]$. The algorithm processes the points as they are stored in the array, i.e., in decreasing $y$-order. It maintains the invariant that, when processing point $\mathtt{A}[j]$, all points that already have been identified as "below $\mathcal{L}_{\kappa-1}$" are stored in decreasing $y$-order in $\mathtt{A}[\ell_b + \kappa, \dots, i-1]$ for some $i \in [\ell_b + \kappa, \dots, j]$—see Figure 1. Since the points are sorted according to $<_y$, we can efficiently sweep the plane top-down. For each $h \in [0, \dots, \kappa-1]$, we maintain the *tail* $\tau_h$, the lowest point seen so far that is known to lie on $\mathcal{L}_h$.

The point $\mathtt{A}[j]$ now either is classified as "below $\mathcal{L}_{\kappa-1}$" ($p_{i_1}$ in Figure 2) or replaces the tail $\tau_h$ of some layer $\mathcal{L}_h$ ($p_{i_2}$ in in Figure 2). In the former case, $\mathtt{A}[j]$ is stably moved directly behind $\mathtt{A}[\ell_b + \kappa, \dots, i-1]$, i.e., it is swapped with $\mathtt{A}[i]$ and $i$ is then incremented by one. In the latter case, $\mathtt{A}[j]$ is swapped with $\tau_h$, i.e., with $\mathtt{A}[\ell_b + h]$, and we increment the counter $c_h$

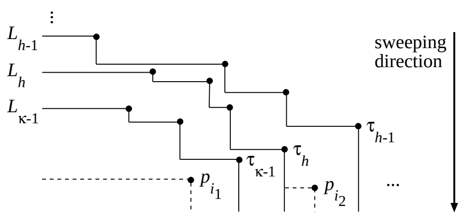


Figure 2: Maintaining the lowest point of each layer.

by one. When we have reached the end of the array, we inductively see that $\mathtt{A}[\ell_b+\kappa,\ldots,i-1]$ contains the points below $\mathcal{L}_{\kappa-1}$ in sorted order. Furthermore, by the definition of $\xi$, we know that the two subarrays $\mathtt{A}[\ell_b,\ldots,\ell_b+\kappa-1]$ (containing the tails) and $\mathtt{A}[i,\ldots,n-1]$ (containing the remaining points on the layers $\mathcal{L}_0$ through $\mathcal{L}_{\kappa-1}$) together consist of exactly $\xi$ points. We then swap (in linear time) $\mathtt{A}[\ell_b+\kappa,\ldots,i-1]$ (containing the elements below $\mathcal{L}_{\kappa-1}$) and $\mathtt{A}[i,\ldots,n-1]$ such that the $\xi$ elements on the layers $\mathcal{L}_0$ through $\mathcal{L}_{\kappa-1}$ (tails and non-tails) are blocked in $\mathtt{A}[\ell_b,\ldots,\ell_b+\xi-1]$. To re-establish the $y$-order of these $\xi$ points, we sort them in $\mathcal{O}(\xi\log_2\xi)\subset\mathcal{O}(\xi\log_2 n)$ time, that is, we establish Invariant (SORT) for $\mathtt{A}[\ell_b,\ldots,\xi-1]$. Thus, the overall running time for counting the number of points on the topmost $\kappa$ layers and for re-establishing Invariant (SORT) is $\mathcal{O}(n+\xi\log_2 n)$ where $\xi=\sum_{i=0}^{\kappa-1}c_i$.

**Sorting the points by layer** Using the counters $c_i$ computed during the previous step, we now run a variant of counting sort to extract the layers $\mathcal{L}_0$ through $\mathcal{L}_{\kappa-1}$ in sorted $y$-order. To do this in-place, we use the subarray $\mathtt{A}[\ell_b+\xi,\ldots,\ell_b+2\xi-1]$ as scratch space that will hold the layers to be constructed (note that we assume $\ell_b+2\xi\leq n$ and that $\xi=\sum_{i=0}^{\kappa-1}c_i$ holds by definition). To re-establish Invariant (SORT), we finally sort $\mathtt{A}[\ell_b+\xi,\ldots,\ell_b+2\xi-1]$ (note that the points $\mathtt{A}[\ell_b+2\xi,\ldots,n-1]$ have not been touched and thus still are sorted) and update $\ell_b:=\ell_b+\xi$. The running time for sorted extraction of the $\xi$ points on the topmost $\kappa$ layers and for re-establishing Invariant (SORT) for $\mathtt{A}[\ell_b+\xi,\ldots,n-1]$ is $\mathcal{O}(n+\xi\log_2 n)$.

## 3.2 Extracting all Layers in Sorted Order

The above algorithm is built on two major assumptions that need to be maintained in an in-place setting: (1) we have to have access to $\kappa$ counters and (2) the subarray $\mathtt{A}[\ell_b,\ldots,n-1]$ has to be large enough to accommodate two subarrays of size $\xi$. The first issue to be resolved is how to maintain a non-constant number $\kappa$ of counters without using $\Theta(\kappa)$ extra space. Each such counter $c_i$ is required to represent values up to $n$, and thus has to consist of $\log_2 n$ bits.

We will resort to a standard technique in the design of space-efficient algorithms, namely to encode a single bit by a permutation of two distinct (but comparable) elements $q$ and $r$: assuming $q<r$, the permutation $rq$ encodes a binary zero, and the permutation $qr$ encodes a binary one. As the elements in our case are two-dimensional points, we will use the $y$-order for deciding whether two points encode a binary zero or a binary one.[1] Using a block of $\frac{1}{3}n$ elements, we can encode $\frac{1}{6}n$ bits, i.e., $\frac{1}{6}n/\log_2 n$ counters, and this implies that the maximum number of layers for which we can run the algorithm of Lemma 4 is $\kappa=\frac{1}{6}n/\log_2 n$. Lemma 4 gives an $\mathcal{O}(n+\xi\log_2 n)$ bound for each run, and thus we have to make sure that maintaining the counters does not interfere with keeping the overall number of iterations in $\mathcal{O}(\log_2 n)$.

### 3.2.1 The case $\ell_b<\frac{1}{3}n$

If, prior to the current iteration, $\ell_b<\frac{1}{3}n$ holds, we maintain the counters in $\mathtt{A}[\frac{2}{3}n,\ldots,n-1]$.

**Counting the points on the topmost $\kappa$ layers** By Invariant (SORT), $\mathtt{A}[\ell_b,\ldots,n-1]$ is sorted by decreasing $y$-coordinate, so all counters encode the value zero. We set $\kappa:=\frac{1}{6}n/\log_2 n$ and count the elements on each of the topmost $\kappa$ layers. Note that, since the algorithm will process *all* points in $\mathtt{A}[\ell_b,\ldots,n-1]$, any point $q$ in $\mathtt{A}[\frac{2}{3}n,\ldots,n-1]$ may be swapped to the front of the array since it may become the tail $\tau_i$ of some layer $\mathcal{L}_i$. Using a more careful implementation of the approach given in Section 3.1, we can compute all counters and re-establish Invariant (SORT) in $\mathcal{O}(n+\xi\log_2 n)$ time. After we have computed the values of all counters $c_i$—but prior to re-establishing Invariant (SORT)—we compute the prefix sums of $c_0$ through $c_{\kappa-1}$, i.e., we replace $c_j$ by $\hat{c}_j:=\sum_{i=0}^{j}c_j$. This can be done in-place spending $\mathcal{O}(\log_2 n)$ time per counter, i.e., in $\mathcal{O}(n)$ overall time. We also maintain the maximal index $\kappa'$ such that $\ell_b+2\hat{c}_{\kappa'}<\frac{2}{3}n$.

**Extracting and sorting the points on the topmost $\kappa$ layers** If the index $\kappa'$ described above exists, we run (a slightly modified implementation of) the algorithm for extracting the $\xi':=\hat{c}_{\kappa'}$ points on the $\kappa'$ topmost layers as described in Section 3.1. Because of the way $\kappa'$ was chosen, we can guarantee that the scratch space of size $\xi'=\hat{c}_{\kappa'}$ needed for the counting-sort-like partitioning will not interfere with the space $\mathtt{A}[\frac{2}{3}n,\ldots,n-1]$ reserved for representing the counters. The total cost for extracting $\xi'$ points is $\mathcal{O}(n+\xi'\log_2 n)$; this also includes the cost for sorting the scratch space $\mathtt{A}[\ell_b+\xi',\ldots,\ell_b+2\xi'-1]$ and re-establishing Invariant (SORT) (see Section 3.1).

[1]A *set* cannot contain duplicates; hence the relative order of two points is unique. Furthermore, the set of maxima of a *multiset* $M$ consists of the same points as the set of maxima of the *set* obtained by removing the duplicates from $M$. Duplicate removal can be done in-place and in $\mathcal{O}(n\log_2 n)$ time.

If $\kappa' < \kappa$, i.e., if we extract some but not all $\kappa = \frac{1}{6}n/\log_2 n$ layers, we will additionally run the $\mathcal{O}(n \log_2 n)$ skyline computation algorithm described in Section 2 as a post-processing step to also extract the points on the next topmost layer, regardless of its size. Similarly, if the index $\kappa'$ does not exist at all, we extract the topmost layer $\mathcal{L}_0$ using the $\mathcal{O}(n \log_2 n)$ skyline computation algorithm on $\mathtt{A}[\ell_b, \ldots, n-1]$—note that in this case the topmost layer $\mathcal{L}_0$ contains $c_0 > \frac{1}{2}\left(\frac{2}{3}n - \ell_b\right) > \frac{1}{2}\left(\frac{2}{3}n - \frac{1}{3}n\right) \in \Theta(n)$ points. In any case, we spend another $\mathcal{O}(n \log_2 n)$ time to re-establish Invariant (SORT) by sorting.

**Analysis** Our analysis classifies each iteration according to whether or not all $\xi$ points on the topmost $\kappa = \frac{1}{6}n/\log_2 n$ layers are moved to their final position in the array. If all $\xi$ points are moved, we know that $\xi \geq \frac{1}{6}n/\log_2 n$, and thus only a logarithmic number of such iterations can exist. Also, we can distribute the $\mathcal{O}(n + \xi \log_2 n)$ time spent per iteration such that each iteration gets charged $\mathcal{O}(n)$ time and that each of the $\xi$ points moved to its final position gets charged $\mathcal{O}(\log_2 n)$ time, so the overall cost for all such iterations is $\mathcal{O}(n \log_2 n)$. If less than $\kappa$ layers can be processed in the iteration in question (this also includes the case that $\kappa'$ does not exist), the $\mathcal{O}(n + \xi \log_2 n)$ cost for counting the $\xi$ points on the topmost $\kappa$ layers and the $\mathcal{O}(n+\xi' \log_2 n)$ cost for extracting $\xi'$ points on the topmost $\kappa'$ layers is dominated by the $\mathcal{O}(n \log_2 n)$ cost for the successive skyline computation. The definition of $\kappa'$ guarantees that, after we have performed the skyline computation, we have advanced the index $\ell_b$ by at least $\frac{1}{2}\left(\frac{2}{3}n - \ell_b\right)$ steps. Since $\ell_b < \frac{1}{3}n$, there is only a constant number of such iterations, hence their overall cost is $\mathcal{O}(n \log_2 n)$.

### 3.2.2 The case $\ell_b \geq \frac{1}{3}n$

If, prior to the current iteration, $\ell_b \geq \frac{1}{3}n$ holds, we maintain the counters in $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$. Note that this subarray contains (part of) the layers that have been computed already. Since maintaining a counter will involve swapping some of the elements in $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$, this will disturb the $y$-order of (some of) the layers already computed, and we have to make sure that we can reconstruct the layer order.

As for the case $\ell_b < \frac{1}{3}n$ we either extract all $\xi$ points on the topmost $\kappa$ layers in $\mathcal{O}(n + \xi \log_2 n)$ time or extract less than $\kappa$ layers followed by a skyline computation in $\mathcal{O}(\nu \log_2 \nu)$ time where $\nu := n - \ell_b$. In both cases, the complexity given also includes the cost for re-establishing Invariant (SORT). Summing up, the cost for all iterations in which $\ell_b < \frac{1}{3}n$ and for all iterations in which $\ell_b \geq \frac{1}{3}n$ is $\mathcal{O}(n \log_2 n)$. Restoring the "counter space" to the original order can be done in the same time complexity, and in the full version, we prove that a simple linear-time scan is sufficient to detect all "layer boundaries". Combining this with the fact that each point gets charged $\mathcal{O}(\log_2 n)$ cost for the iteration in which it is moved to its final location, we obtain our main result:

**Theorem 5** *All layers of maxima of an $n$-element point set in two dimensions can be computed in-place and in optimal time $\mathcal{O}(n \log_2 n)$ such that the points in each layer are sorted by decreasing $y$-coordinate.*

## References

[1] J. Bentley, K. Clarkson, and D. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, 9(2):168–183, 1993.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. 17th Intl. Conf. Data Engineering*, pp. 421–430. 2001.

[3] P. Bose, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and J. Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Computational Geometry: Theory & Applications*, 2006.

[4] H. Brönnimann and T. Chan. Space-efficient algorithms for computing the convex hull of a simple polygonal line in linear time. In *Proc. Latin American Theoretical Informatics*, LNCS 2976, pp. 162–171. 2004.

[5] H. Brönnimann, T. Chan, and E. Chen. Towards in-place geometric algorithms. In *Proc. 20th Symp. Computational Geometry*, pp. 239–246. 2004.

[6] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. Toussaint. Space-efficient planar convex hull algorithms. *Theoretical Computer Science*, 321(1):25–40, 2004.

[7] A. Buchsbaum and M. Goodrich. Three-dimensional layers of maxima. *Algorithmica*, 39(4):275–286, 2004.

[8] H. Dai and X. Zhang. Improved linear expected-time algorithms for computing maxima. In *Proc. Latin American Theoretical Informatics*, LNCS 2976, pp. 181–192. 2004.

[9] V. Geffert, J. Katajainen, and T. Pasanen. Asymptotically efficient in-place merging. *Theoretical Computer Science*, 237(1–2):159–181, 2000.

[10] S. Kapoor. Dynamic maintenance of maxima of 2-D point sets. *SIAM J. Computing*, 29(6):1858–1877, 2000.

[11] H. Kung, F. Luccio, and F. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[12] J. Matoušek. Computing dominances in $E^n$. *Information Processing Letters*, 38(5):277–278, 1991.

[13] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Systems*, 30(1):41–82, 2005.

[14] F. Preparata and M. Shamos. *Computational Geometry. An Introduction.* Springer, 1988.

[15] J. Vahrenhold. Line-segment intersection made in-place. In *Proc. 9th Intl. Workshop Algorithms and Data Structures*, LNCS 3608, pp. 146–157, 2005.

# Finding enclosing boxes with empty intersection

C. Cortés*, J.M. Díaz-Báñez† y J. Urrutia ‡

**Abstract**

Let $S$ be a point set in general position on the plane such that its elements are colored red or blue. We study the following problem: Remove as few points as possible from $S$ such that the remaining points can be enclosed by two isothetic rectangles, one containing all the red points, the other all the blue points, and such that each rectangle contains only points of one color. We prove that this problem can be solved in $O(n^3)$ time and space.

## 1 Introduction

In Data Mining and Classification problems, a natural method to analyse data, is to select prototypes representing different classes of data. A standard technique to achieve this, is to perform cluster analysis on the training data [4, 6]. The clustering can be obtained by using simple geometric shapes such as circles or boxes. In [1, 5], circles and parallel-axis boxes respectively, are considered for the selection. In [1], the following problem is studied: given a bicolored point set, find a ball that contains the maximum number of red points without containing any blue point inside it.

In some cases these methods can produce slanted classifications due to the fact that some data may be defective or contain values out of reasonable ranges. In other cases, we may obtain data hard to classify due to relatively small similarities between different classes. A possible way to find a better classification for the former problem is to remove some data-points from the input. Culling the minimum number of such points can be a suitable criterium to lose as less information as possible. Thus, in this paper we study the following problem: Let $S$ be a bicolored point set in general position on the plane such that no two elements of $S$ lie on a vertical or horizontal line. Find the largest subset $S'$ of $S$ that can be enclosed by two isothetic rectangles $\mathcal{R}$ and $\mathcal{B}$ such that:

- $\mathcal{R}$ (resp. $\mathcal{B}$) contains all the red (resp. blue) points of $S'$ respectively
- $\mathcal{R}$ (resp. $\mathcal{B}$) contains no blue (resp. red) points of $S'$.

We will refer to this problem as *Empty Intersection Enclosing Boxes* problem or simply as *EIEB-problem*.

For example, the solution to the *EIEB-problem* for the point set shown in Figure 1 is 2, since by removing the points $r_1$ and $b_1$ we can obtain two rectangles, $\mathcal{R}$ and $\mathcal{B}$ each of them containing only red and blue points respectively.

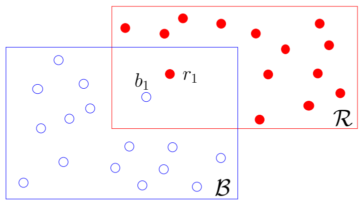


Figure 1: Removing points $r_1$ and $b_1$, we get a solution.

From now on, an isothetic rectangle enclosing a set of red (resp., blue) points will be called *red rectangle*, denoted by $\mathcal{R}$ (resp., *blue rectangle*, denoted by $\mathcal{B}$).

To solve our problem, we observe first that given a bicolored point set $S$, and two rectangles $\mathcal{R}$ and $\mathcal{B}$ that provide an optimal solution to the *EIEB-problem* for $S$, there are three types of relative positions of $\mathcal{R}$ with respect to $\mathcal{B}$, up to symmetry. These are depicted in Figure 2. We call a *corner solution* to that in which $\mathcal{R}$ contains exactly one corner of $\mathcal{B}$; a *sandwich solution* to that in which $\mathcal{R}$ intersects properly two parallel sides of $\mathcal{B}$; and *disjoint solution* to that in which $\mathcal{R}$ and $\mathcal{B}$ can be separated either by a horizontal or a vertical line.

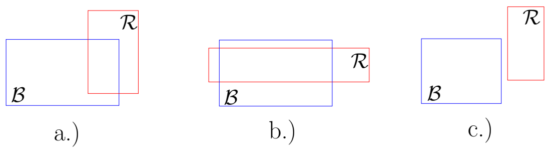


Figure 2: a.) Corner, b.) sandwich and, c.) disjoint solutions.

Our procedure consists on looking for the best solution of each type keeping the best among them. In all of the previous cases, we reduce our 2-dimensional problem to the following 1-dimensional problem:

***Maximum Consecutive Subsequence (MCS):*** Given a sequence $x_1, x_2, ..., x_n$ of $0's$, $+1$'s and $-1's$,

*Departamento Matemática Aplicada I, Universidad de Sevilla, ccortes@us.es

†Departamento Matemática Aplicada II, Universidad de Sevilla, dbanez@us.es

‡Instituto de Matemáticas, Universidad Nacional Autónoma de México, urrutia@matem.unam.mx

compute, for every index $i = 1, ..., n$, all the subsequences $x_i, x_{i+1}, ..., x_j$ of consecutive elements such that $x_i + x_{i+1} + \ldots + x_j$ is maximized over all subsequences starting at $x_i$.

It is relatively easy to see that the *MCS-problem* can be solved in linear time by using the same techniques used to solve Bentley's *Maximum segment sum* problem [2].

## 2 Finding the optimum Corner Solution

We now sketch the key idea to find the best *corner solution.* The remaining cases are solved using similar techniques.

Let $(\mathcal{R}, \mathcal{B})$ be a corner type pair of rectangles. Assume for the rest of this section that, as in Figure 3, $\mathcal{R}$ contains the topmost right corner of $\mathcal{B}$. We denote by $Red(\mathcal{R} \setminus \mathcal{B})$ to the set of red points of $S$ contained in $\mathcal{R} \setminus \mathcal{B}$. Similarly we define $Blue(\mathcal{R} \setminus \mathcal{B})$, $Red(\mathcal{B} \setminus \mathcal{R})$, $Blue(\mathcal{B} \setminus \mathcal{R})$, $Red(\mathcal{R} \cap \mathcal{B})$, and $Blue(\mathcal{R} \cap \mathcal{B})$, see Figure 3. We remark that $\mathcal{R}$ and $\mathcal{B}$ are considered to be closed sets.

**Proposition 1** *Let $(\mathcal{R}, \mathcal{B})$ be a corner type pair of rectangles. Then, it is possible to find another corner type pair $(\hat{\mathcal{R}}, \hat{\mathcal{B}})$ such that $\hat{\mathcal{R}} \setminus \hat{\mathcal{B}}$ (resp., $\hat{\mathcal{B}} \setminus \hat{\mathcal{R}}$) contains at least $Red(\mathcal{R} \setminus \mathcal{B})$ red points (resp., $Blue(\mathcal{B} \setminus \mathcal{R})$ blue points) and the sides of $\hat{\mathcal{R}}$ (resp., $\hat{\mathcal{B}}$) go through red (resp., blue) points.*

**Corollary 2** *There exists a pair $(\mathcal{R}, \mathcal{B})$ of corner type rectangles that provides an optimal* corner solution *such that the sides of $\mathcal{R}$ (resp., $\mathcal{B}$) go through red (resp., blue) points.*

From now on, any rectangle $\mathcal{R}$ (resp., $\mathcal{B}$) will be considered to be delimited by red (resp., blue) points of $S$.

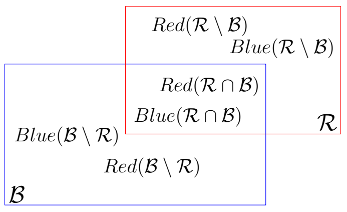
A pair $(\mathcal{R}, \mathcal{B})$ of corner type rectangles that provides an optimal *corner solution* will be a pair that maximizes the sum $|Red(\mathcal{R} \setminus \mathcal{B})| + |Blue(\mathcal{B} \setminus \mathcal{R})|$.

Let $\mathcal{Q}_\mathcal{R}$ be the quadrant obtained from $\mathcal{R}$ by extending to infinity its left and lower sides towards the North and the East respectively. We will refer to $\mathcal{Q}_\mathcal{R}$ as the *red quadrant.* Similarly we define the *blue quadrant* $\mathcal{Q}_\mathcal{B}$, obtained by extending to infinity the right and upper sides of $\mathcal{B}$ towards the South and West respectively. We will assume that the quadrants include their borders.

As a consequence of Proposition 1, if $(\mathcal{R}, \mathcal{B})$ provides an optimal solution, then $|Red(\mathcal{R} \setminus \mathcal{B})| = |Red(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})|$ and $|Blue(\mathcal{R} \setminus \mathcal{B})| = |Blue(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})|$. We then reformulate our problem as follows: find the pair of quadrants $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ that maximize the sum $|Red(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| + |Blue(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})|$.

It is easy to see that by using *range search* techniques [3], we can solve this problem in $O(n^4)$ time.

We proceed now to show how to solve the *EIEB-problem* in $O(n^3)$ time.



$Red(\mathcal{R} \setminus \mathcal{B})$
$Blue(\mathcal{R} \setminus \mathcal{B})$
$Red(\mathcal{R} \cap \mathcal{B})$
$Blue(\mathcal{R} \cap \mathcal{B})$
$\mathcal{R}$
$Blue(\mathcal{B} \setminus \mathcal{R})$
$Red(\mathcal{B} \setminus \mathcal{R})$
$\mathcal{B}$

Figure 3: Notation for points in $S$ depending on their location.

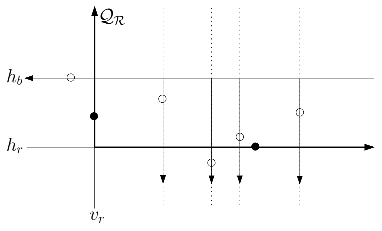


Figure 4: Illustrating the Query problem.

### 2.1 The Algorithm

Consider the orthogonal grid generated by drawing horizontal and vertical lines through the elements of $S$.

Assume we have already preprocessed points in $S$ so that it is possible to answer in $O(1)$ *amortized time* the next problem (Figure 4):

**OptQuad:** Given a red quadrant $\mathcal{Q}_\mathcal{R}$, determined by two red lines $< h_r, v_r >$, and a blue horizontal line $h_b$ above $h_r$, find the blue vertical line $v_b$ to the right of $v_r$ such the blue quadrant $\mathcal{Q}_\mathcal{B}$ bounded above and to its right by $h_b$ and $v_r$ respectively, is such that it maximizes the sum $|Red(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| + |Blue(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})|$ over all the possible choices of $v_b$.

Our algorithm to solve an instance of a *corner type* solution proceeds as follows.

***Corner Solution* Algorithm (CS-Algorithm):**

1. For each vertex of the grid, compute the number of red and blue points of $S$ laying in the four (North-West, North-East, South-West and South East) quadrants with vertex in it.
2. For each red quadrant $\mathcal{Q}_\mathcal{R} < h_r, v_r >$ and for every blue horizontal line $h_b$ above $h_r$, the query reports a blue vertical line $v_b$. Store the sum $s_{(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})} = |Red(\mathcal{Q}_\mathcal{R} \setminus \mathcal{Q}_\mathcal{B})| + |Blue(\mathcal{Q}_\mathcal{B} \setminus \mathcal{Q}_\mathcal{R})|$.
3. Output the pair $(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})$ that provides the maximum value $s_{(\mathcal{Q}_\mathcal{R}, \mathcal{Q}_\mathcal{B})}$.

**Complexity**: It is easy to see that the first step of our algorithm can be completed in quadratic time [3]. The second step of our algorithm answers $O(n^3)$ queries. These queries can be solved in amortized constant time using **OptQuad**. Finally reporting the best $s_{(\mathcal{Q}_\mathcal{R},\mathcal{Q}_\mathcal{B})}$ can be done in constant time.

## 2.2 The Preprocessing

We now describe briefly the preprocessing needed to solve **OptQuad** and prove the correctness of the whole algorithm. Consider the orthogonal grid obtained by passing a horizontal and a vertical line through every element of $S$. Assume that these lines are colored red or blue according to the color of the point in $S$ they contain.

Each pair consisting of a horizontal blue line $h_b$ and a horizontal red line $h_r$ below it determines a horizontal strip $HS_{h_b,h_r}$. We assign weights to some elements of $S$ according to the following criteria, see Figure 5 a.):

1. Every red point inside $HS_{h_b,h_r}$ has weight $-1$
2. Red points in or above $HS_{h_b,h_r}$ have weight 0
3. Blue points in $HS_{h_b,h_r}$ have weight 0
4. Blue points below $HS_{h_b,h_r}$ have weight $+1$

Blue points above $HS_{h_b,h_r}$ and red points below $HS_{h_b,h_r}$ are discarded. We next project our blue and red points together with their weights on the $x-axis$ obtaining a sequence $\mathcal{P} = \{p_{\sigma(1)}, \ldots, p_{\sigma(k)}\}$ of points with weights $-1, 0$, or $+1$, where $k$ is the number of weighted points of $S$. We use now the solution to the *MCS* problem, and find for each $p_{\sigma(i)}$ the $j > i$ s.t. the sum of the weights of all the elements in $\mathcal{P}$ between $p_{\sigma(i)}$ and $p_{\sigma(j)}$ (including the weight of $p_{\sigma(j)}$) is maximized.

Suppose now that we have a red quadrant bounded below by $h_r$ and to its left by a vertical line $v_r$ through a red point above $h_r$, and a blue quadrant bounded above by $h_b$ and to its right by a vertical line $v_b$ through a blue point as shown in Figure 5 b.). Let $\mathcal{Q}'$ be the quadrant bounded above by $h_b$, and to the right by $v_r$.

Let us define the following numbers:

- Let $b_1$ be the number of blue points in $\mathcal{Q}'$
- Let $r_1$ be the number of red points in $\mathcal{Q}_\mathcal{R}$
- Let $c$ be the sum of the weights of the elements of $\mathcal{P} = \{p_{\sigma(1)}, \ldots, p_{\sigma(k)}\}$ between $p_{\sigma(i)}$ and $p_{\sigma(j)}$, where $p_{\sigma(i)}$ and $p_{\sigma(j)}$ are the points into which points in $v_r$ and $v_b$ were projected in $\mathcal{P} = \{p_{\sigma(1)}, \ldots, p_{\sigma(k)}\}$.

**Theorem 3** *The number of red points in $\mathcal{Q}_\mathcal{R}$ plus the number of blue points in $\mathcal{Q}_\mathcal{B}$ minus the number of blue and red points in $\mathcal{Q}_\mathcal{R} \cap \mathcal{Q}_\mathcal{R}$ equals $b_1 + r_1 + c$.*

It follows now that we have to maximize $c$ to find the optimal solution in which $\mathcal{Q}_\mathcal{R}$ participates, and $\mathcal{Q}_\mathcal{B}$ is bounded above by $h_b$. This can be done using the solution to the *SMC problem* in $\mathcal{P} = \{p_{\sigma(1)}, \ldots, p_{\sigma(k)}\}$. Thus we have:

**Theorem 4** *An optimum corner solution can be found in $O(n^3)$ time and storage, given $O(n^3)$ preprocessing time.*

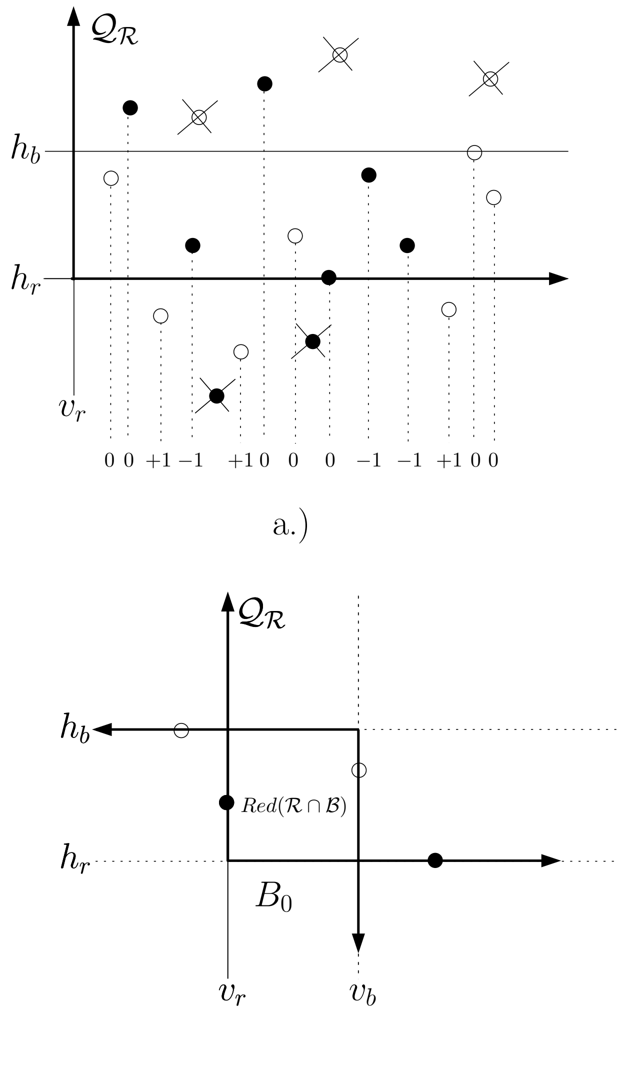

$\mathcal{Q}_\mathcal{R}$
$h_b$
$h_r$
$v_r$
0 0 +1 −1 +1 0 0 0 −1 −1 +1 0 0
a.)
$\mathcal{Q}_\mathcal{R}$
$h_b$
$Red(\mathcal{R} \cap \mathcal{B})$
$h_r$
$B_0$
$v_r$
$v_b$
b.)

Figure 5: Illustrating the proof of the algorithm's correctness.

## 3 Conclusions

We propose an algorithm to solve the *EIEB-problem* that requires $O(n^3)$ time and quadratic space. The algorithm solves separately three different possibilities for an optimal solution type: *corner*, *sandwich* and, *disjoint solution.* We have presented here the way to solve the optimum *corner solution.* This is the most interesting case, the others can be solved in a similar way.

To conclude, let us mention that the 1-dimensional EIEB-problem, where a set of red and blue points on a line are given and we are asked to determine the minimum number of points to be removed in order to get two disjoint intervals containing points of only one color, can be solved in lineal time.

## References

[1] B. Aronov, S. Har-Peled. On approximating the depth and related problems. *Proceedings 16th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2005), 2005.

[2] J. Bentley, “Programming pearls: algorithm design techniques,” Comm. ACM, vol. 27, no. 9, pp. 865 - 873, 1984.

[3] J.L. Bentley and M.I. Shamos, *A problem in multivariate statistics: Algorithms, data structure and applications*, Proceedings of the 15th annual Allerton Conference on Communications, Control, and Computing, pp. 193-201, 1977.

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

[5] Eckstein, J., Hammer, P.L., Liu, Y., Nediak,M., Simeone, B. The maximum box problem and its applications to data analysis. Comput. Optim. Appl. 23, 2002, 285-298.

[6] Hand,H., Mannila, H., Smyth, P. Principles of Data Mining. The MIT Press, 2001.

# Inner Approximation of Polygons and Polyhedra by Unions of Boxes

Christian Spielberger[*] Martin Held[†]

## Abstract

Given a multiply-connected polygonal area $\mathcal{P}$ in the plane and a point set $S \subset \mathbb{R}^2$, where some points of $S$ may lie inside of $\mathcal{P}$, we present a fast approximation method for finding a largest axis-aligned or oriented rectangle contained in $\mathcal{P}$ which does not contain any points of $S$. All standard meanings of "largest" are supported, such as maximum area and maximum perimeter. This heuristic is extended to finding $k$ rectangles whose union is largest. Furthermore, we present an extension of our method to 3D, i.e., to computing inner approximations of polyhedra (possibly with holes, voids and cavities) by unions of (oriented) boxes.

Our 2D algorithm is based on a discretization of space by means of a regular mesh of size $w \times h$ and on a discretization of the rotation angles. Let $n$ be the sum of the number of vertices of $\mathcal{P}$ and the number of points in $S$. Then an inner approximation by $k$ rectangles is found in $O\left(mn(w+h) + k2^k(mwh)^k\right)$ time and $O(wh+n)$ space, where $m$ denotes the number of rotation angles tested. A similar bound is obtained for the 3D case. Several algorithmic improvements help to decrease the time complexity in practice considerably, thus making it quite feasible to determine inner approximations of complex objects by several boxes within a few seconds of CPU time. Extensive practical tests have yielded a formula for predicting a mesh resolution suitable for achieving the approximation quality sought by a user.

## 1 Introduction

**Motivation.** A problem that often arises in metal or textile industry is to cut a rectangle as large as possible out of an arbitrarily shaped piece of sheet metal or cloth [3]. Typically, the term "large" means "maximum area" but other measures of size (such as maximum perimeter) may also be of interest. Sometimes the situation is further complicated if discrete spots of material imperfectness are to be excluded from the rectangle sought.

Finding one or more maximal boxes that are located inside of a shape can also be regarded as an *inner approximation* (or *inner cover*) of that shape. For instance, for visibility tests we are asked whether an object $\mathcal{O}$ in the foreground occludes other objects in the background. Since visibility tests in 3D scenes are very time consuming for complex objects $\mathcal{O}$, researchers in graphics have long been interested in inner approximations of $\mathcal{O}$ by one (or more) convex shapes, such boxes, spheres and ellipsoids: rather than testing a 3D scene for occlusion against $\mathcal{O}$, it may be significantly faster to test the scene against an inner approximation of $\mathcal{O}$. Similarly, simple pre-tests for path planning are based on inner approximations.

**Overview of Results.** We study the following problem: Given an integer $k$, a multiply-connected planar area $\mathcal{A}$ and a set $S$ of $n$ points (in $\mathbb{R}^2$), find $k$ rectangles inside of $\mathcal{A}$ such that the rectangles do not contain any point of $S$ and such that their union has maximum size according to a measure $\mu$. Typically, $\mu$ will denote the area of a rectangle but our algorithm will be able to deal with any measure $\mu$ provided that $\mu(A) \leq \mu(B)$ if $A \subseteq B$ for two planar areas $A$ and $B$. In particular, $\mu$ could also measure the perimeter or the length of a rectangle. Depending on the application, the rectangles will have sides parallel to the coordinate axes (*axis-aligned*) or they will be *oriented* rectangles, i.e., they will have arbitrary orientations.

A natural extension to 3D asks to determine $k$ cuboids inside of a polyhedron $\mathcal{P}$ such that the size of their union is maximum with respect to $\mu$ and such that no point of a set $S$ of points of $\mathbb{R}^3$ is contained in a cuboid. As for the 2D polygons, the polyhedron $\mathcal{P}$ may contain voids, holes and cavities. Again, we are interested in both axis-aligned and oriented cuboids.

In the sequel we present a fast heuristic for finding such a set of $k$ large boxes in 2D and 3D. Our algorithm is based on a discretization of space by means of a regular mesh of size $w \times h$. Furthermore, we apply a straightforward discretization of the rotation angles. Let $n$ be the sum of the number of vertices of $\mathcal{P}$ and the number of points in $S$. Then the worst-case time complexity of the 2D algorithm is $O\left(mn(w+h) + k2^k(mwh)^k\right)$, where $m$ denotes the number of rotation angles tested; its space complexity is $O(wh+n)$. A similar bound is obtained for the 3D case. Several algorithmic improvements help to decrease the time complexity in practice considerably, thus making it quite feasible to determine inner approximations of complex objects by several axis-

[*]Department of Scientific Computing, University of Salzburg, Salzburg, Austria; `christian.spielberger@aon.at`

[†]Department of Scientific Computing, University of Salzburg, Salzburg, Austria; `held@cosy.sbg.ac.at`

aligned or 2–3 oriented boxes within a few seconds.

Since our algorithm depends on the resolution of the mesh it cannot guarantee a constant-factor approximation of the true maximum size of an optimum set of $k$ boxes. Furthermore, there is an obvious trade-off between the quality of the inner approximation and the resolution of the mesh. Based on extensive practical tests we came up with a formula that allows to predict a mesh resolution such that the approximation quality sought by a user can be expected to be achieved with a user-specified probability.

Our algorithms for inner approximation by $k$ boxes have been implemented in C++ for both the 2D and the 3D case. In the sequel, we will mostly focus on the 2D case, though.

**Related Work.** Restricted versions of our problem have received considerable interest in the past. For the 2D case, if $\mathcal{P}$ is restricted to a rectangle $A$, the problem of finding the largest rectangle inside of $A$ whose sides are parallel with those of $A$ and which does not contain a point of $S$ was discussed in a variety of papers, see [8] for a survey. (In most papers, "largest" means "largest area".) The fastest algorithm that solves this problem runs in $O(n \log n + s)$ time, where $s$ is the number of axis-aligned rectangles, whose edges contain a point of $S$ or are contained in the border of $A$ [8]; its expected time complexity is $O(n \log n)$. It is notable that this algorithm uses only $O(n)$ memory. A more general problem is to find the largest empty oriented rectangle bounded by a point of $S$ on each of its four sides. This problem can be solved in $O(n^3)$ time [2, 6].

A largest-area axis-aligned rectangle (LAR) inside of a multiply-connected polygonal area can be found in $O(n \log^2 n)$ time [3]. For polygons without islands an $O(n \log n)$ algorithm is presented in [1]. Note that none of those papers considers additional point constraints within $\mathcal{P}$. (That is, the set $S$ is empty.) Also, nothing is known for finding $k$ (oriented) rectangles (LORs) such that their union has maximum size.

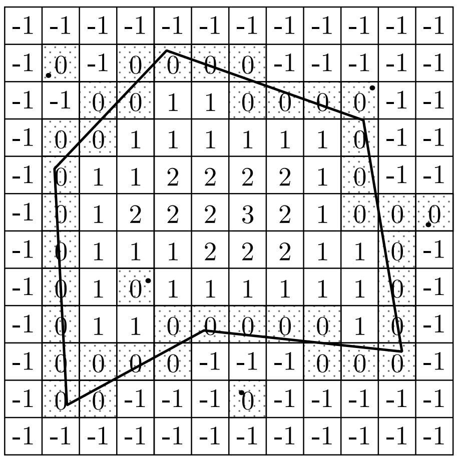
For the 3D case of a set $S$ of $n$ points inside of a bounding cuboid, a largest empty cuboid can be computed in $O(n^3)$ time and $O(n^2 \log n)$ space in the worst case, see [4]. In [7], it is claimed that all locally maximal cuboids can be reported in $O(c + n^2 \log n)$ time and $O(n)$ space, where $c$ is bound by $O(n^3)$.

No algorithms are known for finding largest axis-aligned and largest oriented cuboids inside a polyhedron. As in 2D, an inner approximation by $k > 1$ axis-aligned or oriented boxes also is an open problem.

## 2 Finding a LAR Inside a Polygon

Our heuristic for computing a maximum axis-aligned rectangle (relative to the measure $\mu$) uses a regular mesh for discretizing the plane. A rectangle that consists entirely of mesh cells is called a *mesh rectangle*. Our algorithm finds the largest mesh rectangle which lies completely in the given polygon $\mathcal{P}$ and does not contain any point of $S$.

Consider a regular mesh $\mathcal{M} := \{0, 1, \ldots, w-1\} \times \{0, 1, \ldots, h-1\}$ with resolution $w \times h$. In the first step, all cells $\mathcal{M}(a, b)$ that are intersected by the boundary $\partial\mathcal{P}$ of the polygon are determined and their cell values are set to zero. Then all cells that include at least one point of $S$ are set to zero. Both can be done in $O(n(w + h))$ time. Then all outer cells are set to $-1$ in $O(wh)$ time. The inner cells are set to the values specified by the *chessboard distance*. Several algorithms are known for computing the chessboard distance on a $w \times h$ mesh in $O(wh)$ time, see, for instance, [5]. We use a modified flood-fill algorithm: in the first step, all eight neighbor cells of zero cells that currently have undefined values get the value one. Then the neighbor cells of 1-cells get the value 2, and so on. (In Fig. 1, the shaded cells depict the zero cells occupied by $\partial\mathcal{P}$ and the points of $S$). The resulting value obtained for a cell $\mathcal{M}(a, b)$ is denoted by $d(a, b)$.



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | -1 | -1 |
| -1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | -1 | -1 |
| -1 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 0 | -1 | -1 |
| -1 | 0 | 1 | 2 | 2 | 2 | 3 | 2 | 1 | 0 | 0 | 0 |
| -1 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 0 | -1 |
| -1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | -1 |
| -1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | -1 |
| -1 | 0 | 0 | 0 | 0 | -1 | -1 | -1 | 0 | 0 | 0 | -1 |
| -1 | 0 | 0 | -1 | -1 | -1 | 0 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Figure 1: Discretization of a polygon.

The *square anchored at* cell $\mathcal{M}(a, b)$ is defined as the square with lower left cell $\mathcal{M}(a-(d-1), b-(d-1))$ and upper right cell $\mathcal{M}(a+(d-1), b+(d-1))$, where $d = d(a, b)$. It is easy to see that every such square is fully contained in $\mathcal{P} \backslash S$.

For an inner cell $\mathcal{M}(a, b)$ the number of cells which are to the right of it and which have a cell value of at least $d(a, b)$ is given by the *horizontal length code* $h(a, b)$. Similarly, a *vertical length code* $v(a, b)$ is defined for each inner cell. The horizontal length code for each cell of a row $\mathcal{M}(., b)$ can be set in $O(w)$ time by stepping from cell $\mathcal{M}(w-1, b)$ to cell $\mathcal{M}(0, b)$ and by performing the following tasks for each inner cell $\mathcal{M}(a, b)$.

- If $d(a, b) > d(a+1, b)$ then the length code $h(a, b)$

stays 0.

- If $d(a,b) = d(a+1,b)$ then the length code $h(a,b)$ is set to $h(a+1,b)+1$.
- If $d(a,b) < d(a+1,b)$ then the length code is set to the value $h(a+1,b)+h(c,b)+2$, where $c = \min\{x \in \mathbb{N} : x > a \text{ and } d(x,b) = d(a,b)\}$.

Suppose $d = d(a,b)$. The value $h(c,b)$ is the length code of the next $d$-cell in this row. This value is stored in an array with index $d$. Thus, it can be retrieved in constant time. These tasks are done for each row in the mesh. The vertical length code is set in a similar manner, by stepping downwards through the cells of columns. The calculation of the length code takes $O(wh)$ time.

The mesh rectangle with lower-left cell $\mathcal{M}(l_a, l_b)$ and upper-right cell $\mathcal{M}(u_a, u_b)$ is denoted by $R[(l_a, l_b), (u_a, u_b)]$. For any inner cell $\mathcal{M}(a,b)$ the mesh rectangle $\text{rec}_{h,1}(a,b) := R[(a-d+1, b-d+1), (a+h+d-1, b+d-1)]$, where $d = d(a,b)$ and $h = h(a,b)$, is the *horizontal mesh rectangle anchored at* cell $\mathcal{M}(a,b)$ with a *one-row core.* If $d(a,b+1) \geq d(a,b) \geq 1$, the mesh rectangle $\text{rec}_{h,2}(a,b) := R[(a-d+1, b-d+1), (a+h_2+d-1, b+d)]$, where $d = d(a,b)$ and $h_2 = \min(h(a,b), h(a,b+1))$, is the *horizontal mesh rectangle anchored at* cell $\mathcal{M}(a,b)$ with a *two-row core.* Similarly, *vertical mesh rectangles* with *one-* and *two-column cores* are defined. Since an arbitrarily anchored mesh rectangle can be covered by a set of anchored squares, it is fully contained in $\mathcal{P} \backslash S$.

**Theorem 1** *For an arbitrary mesh rectangle $R$ in $\mathcal{P}$ that does not contain any point of $S$ there is an anchored mesh rectangle which covers $R$.*

**Theorem 2** *Given an arbitrary $n_1$-vertex polygon $\mathcal{P}$ in the plane, a point set $S$ with $n_2$ points, and a mesh $\mathcal{M}$ with resolution $w \times h$ covering $\mathcal{P}$, a largest mesh rectangle contained in $\mathcal{P} \backslash S$ can be found in $O(n(w+h)+wh)$ time using $O(wh+n)$ memory, where $n = n_1 + n_2$.*

**Proof.** Setting the zero cells and the outer cells can be done in $O(n(w+h)+wh)$ time and $O(wh+n)$ space. The cell values in the interior of $\mathcal{P}$ can be calculated without further costs. The length code can also be set in $O(wh)$ time for the whole mesh.

Let $S'$ be the set of all anchored mesh rectangles. Due to Theorem 1, the set $S'$ is a sufficient search space for the largest mesh rectangle inside $\mathcal{P}$. Since there are only $O(wh)$ mesh rectangles in $S'$, for each inner cell one, searching $S'$ does not increase the time and space complexity further. □

An inner cell $\mathcal{M}(a,b)$ is called a *horizontal upward cell,* if $d(a,b) > d(a-1,b)$ or
$d(a,b) = d(a,b+1)$ and $d(a,b) > d(a-1,b+1)$.

Similarly, an inner cell $\mathcal{M}(a,b)$ is called a *vertical upward cell,* if $d(a,b) > d(a,b-1)$ or
$d(a,b) = d(a+1,b)$ and $d(a,b) > d(a+1,b-1)$.

A mesh rectangle which is anchored in an upward cell is called *upward anchored.*

**Theorem 3** *For any anchored mesh rectangle $R$ there is an upward anchored mesh rectangle which covers $R$.*

Thus, the search space can be reduced to the set of all upward anchored mesh rectangles. Each upward cell gets a pointer to the next upward cell for this reason. The first cell in a mesh row (column) gets the pointer to the first upward cell in this row (column). Hence, many cells can be left out during the search. This does not reduce the worst-case time complexity but tends to decrease the practical run-time of the program considerably.

## 3 General Orientation

By using the method of Section 2 a simple heuristic for the LOR can be built. Again, given is an arbitrary polygon $\mathcal{P}$ and a point set $S$. The angle range $[0 \mathinner{..} \frac{\pi}{2}]$ is discretized. The input data is rotated by the angles $0, \delta, 2\delta, \ldots, (m-1)\delta$, where $m$ is an integer with $m \geq 2$, and $\delta := \frac{\pi}{2m}$. The LAR is calculated for each of the rotated data sets. The largest mesh rectangle returned by the LAR algorithm is the result of the LOR algorithm. (Of course, this mesh rectangle has to be re-transformed to the original orientation.)

## 4 Generalization to 3D Space

We extended the described method to approximate the maximum volume cuboid inside a given polyhedron $\mathcal{P}$ such that no point of $S$ is included. A 3D mesh was used to discretize the space. However, the length code can not be adopted directly to the 3D mesh. Instead the length codes have to be calculated for each of the three coordinate planes $xy$, $xz$ and $yz$. In addition a 3D chessboard distance is used to retrieve the maximal thickness of each cuboid.

## 5 Inner Approximation by Several Objects

Let $\mathcal{F}$ be a finite set of shapes in $\mathbb{R}^d$. (In our application, $d = 2, 3$.) Let $k \geq 2$ be an integer. Given a region $\mathcal{R} \subset \mathbb{R}^d$ and a set of points $S \subset \mathbb{R}^d$, we want to find $k$ shapes $C_1, \ldots, C_k \in \mathcal{F}$ such that $\cup_{1 \leq i \leq k} C_i \subseteq \mathcal{R} \setminus S$ and $\mu(\cup_{1 \leq i \leq k} C_i)$ is maximum. Note that the shapes $C_i$ may intersect. Let $\binom{\mathcal{F}}{k}$ be the set of all $k$-elemental subsets of $\mathcal{F}$. The proposed algorithm performs a brute-force search of the set $\binom{\mathcal{F}}{k}$ for the largest union of $k$ shapes relative to the measure $\mu$. Let $s$ be the cardinality of $\mathcal{F}$. The algorithm has to check $\binom{s}{k}$ shape combinations by a $k$-times nested loop. Thus the union of $k$ shapes has to be calculated $O(s^k)$ times.

The calculation of $\mu$ for such a union is done by the inclusion-exclusion principle, by relying on $\sum_{i=1}^{k}\binom{k}{i} = 2^k$ calculations of $\mu\left(\cap_{C\in\mathcal{B}}C\right)$, where $\mathcal{B}$ can have at most $k$ elements. How long it takes to calculate the intersection of $k$ shapes depends on the complexity of the shapes and has to be analyzed for each specific type of shapes. For example, computing the common intersection of $k$ axis-aligned rectangles takes $O(k)$ time. Thus the calculation of the union volume of $k$ axis-aligned rectangles needs $O\left(k\,2^k\right)$ time, and our shape selector needs at most $O\left(k\,2^k s^k\right)$ time in total.

Two improvements may reduce the average time complexity drastically. First, let $\mathcal{F}'$ be the set of all shapes in $\mathcal{F}$ that are not contained in another shape in $\mathcal{F}$. The shape set $\mathcal{F}'$ is a sufficient search space for the shape combination that maximizes $\mu$. The second improvement relies on $\mathcal{F}'$ being arranged in decreasing order according to $\mu$. Our shape selector uses a $k$-times nested loop to determine $C_1,\dots,C_k$, starting with the the $j$-largest shape in loop $j$. During each pass through the body of a loop the next smaller shape is tested. Suppose that the current candidate shapes selected in loops 1 to $(j-1)$ are $C_{i_1}, C_{i_2}, \dots, C_{i_{j-1}}$, and we are to select a shape in loop $j$. Let $C_{i_j}$ be the next smallest shape after $C_{i_{j-1}}$ in the ordered list of shapes. If

$$M > \mu(C_{i_1}\cup C_{i_2}\cup\cdots\cup C_{i_{j-1}}) + (k-j+1)*\mu(C_{i_j}),$$

where $M$ is the size of the union of the best selection of shapes obtained so far, then an early termination of the loops $j$ to $k$ is possible (since no better result will be obtained) and the next smaller shape is tested in loop $j-1$.

## 6 Choosing a Suitable Mesh Resolution

Let $A$ be the area of the true LAR and let $A'$ be the area of the largest mesh rectangle obtained by our algorithm. The ratio $\alpha = A'/A$ is called *approximation ratio* and should be as close to 1 as possible. Let $B$ be the bounding box of $\mathcal{P}$. The ratio $a_{\mathcal{P}} = \frac{a(\mathcal{P})}{a(B)}$ is called the *relative area* of $\mathcal{P}$, where $a$ is a measure for the area. Let $p(\mathcal{P})$ be the perimeter of $\mathcal{P}$. Further, let $p(B)$ be the perimeter of the bounding box of $\mathcal{P}$. The ratio $p_{\mathcal{P}} = \frac{p(\mathcal{P})}{p(B)}$ is called *relative perimeter* of $\mathcal{P}$. Furthermore, we call the ratio $t_{\mathcal{P}} = a_{\mathcal{P}}/p_{\mathcal{P}}$ the *thickness* of $\mathcal{P}$. It is an attempt to cast an intuitive understanding of the “thickness” of a polygon into a numerical number.

Tests indicated that the higher the thickness of a polygon is the lower the resolution of the mesh can be in order to achieve the same approximation ratio. Let $w\times w$ be the resolution of the mesh $\mathcal{M}$ used. The value $w^2\,t_{\mathcal{P}}$ is called *weighted resolution* of $\mathcal{M}$ relative to $\mathcal{P}$. Since the number of inner cells grows with the thickness, the approximation ratio grows with the weighted resolution. Our 2D algorithm was tested over a variety of polygons with different mesh resolutions. As a result we obtained a look-up table and a formula for the mesh resolution. The parameters for the formula are the weighted resolution, which can be read off from the table, and the area and the perimeter of the polygon. The row index of the table is the desired approximation ratio $\alpha$ and the column index is the probability that $\alpha$ will be achieved.

## 7 Conclusion

We use a regular mesh to discretize 2D and 3D space for computing inner approximations of polygonal/polyhedral shapes by one or more axis-aligned or oriented boxes. The time consumption of the algorithm depends on the mesh resolution which, on the other hand, influences the quality of the approximation obtained. Although the basic idea is rather simple, several algorithmic improvements make our approach quite feasible for the approximation of complex shapes by several boxes. Based on extensive tests we have come up with a heuristic to predict a suitable resolution of the mesh relative to the approximation quality requested by a user.

## References

[1] R. P. Boland and J. Urrutia. Finding the Largest Axis-aligned Rectangle in a Polygon in $O(n\log n)$ Time. In *Proc. 13th Canad. Conf. Comput. Geom.*, pages 41–44, 2001.

[2] J. Chaudhuri, S. Nandy, and S. Das. Largest Empty Rectangle Among a Point Set. *J. Algorithms*, 46(1):54–78, 2003.

[3] K. Daniels, V. Milenkovic, and D. Roth. Finding the Largest Area Axis-Parallel Rectangle in a Polygon. *Comput. Geom. Theory and Appl.*, 7:125–148, 1997.

[4] A. Datta and S. Soundaralakshmi. An Efficient Algorithm for Computing the Maximum Empty Rectangle in Three Dimensions. *Informat. Sciences Appl. An Int. J.*, 128(1):43–65, 2000.

[5] T. Hirata. A Unified Linear-Time Algorithm for Computing Distance Maps. *Inform. Process. Lett.*, 58(3):129–133, May 1996.

[6] A. Mukhopadhyay and S. V. Rao. On Computing a Largest Empty Arbitrarily Oriented Rectangle. *Internat. J. Comput. Geom. Appl.*, 13(3):257–271, 2003.

[7] S. C. Nandy and B. B. Bhattacharya. Maximum Empty Cuboid Among Points and Blocks. *Computers & Math. with Appl.*, 36(3):11–20, 1998.

[8] M. Orlowski. A New Algorithm for the Largest Empty Rectangle Problem. *Algorithmica*, 5:65–73, 1990.

# On the Bounding Boxes Obtained by Principal Component Analysis

Darko Dimitrov[*] Christian Knauer[*] Klaus Kriegel[*] Günter Rote[*]

**Abstract**

Principle component analysis (PCA) is commonly used to compute a bounding box of a point set in $\mathbb{R}^d$. In this paper we give bounds on the approximation factor of PCA bounding boxes of convex polygons in $\mathbb{R}^2$ (lower and upper bounds) and convex polyhedra in $\mathbb{R}^3$ (lower bound).

## 1 Introduction

Substituting sets of points or complex geometric shapes with their bounding boxes is motivated with many applications. For example, in computer graphics, it is used to maintain hierarchical data structures for fast rendering of a scene or for collision detection. Additional applications include those in shape analysis and shape simplification, or in statistics, for storing and performing range-search queries on a large database of samples.

Computing a minimum-area bounding box of a set of $n$ points in $\mathbb{R}^2$ can be done in $O(n \log n)$ time, for example with the rotating caliper algorithm [9]. O'Rourke [6] presented a deterministic algorithm, a rotating caliper variant in $\mathbb{R}^3$, for computing the exact minimum-volume bounding box of a set of $n$ points in $\mathbb{R}^3$. His algorithm requires $O(n^3)$ time and $O(n)$ space. Barequet and Har-Peled [2] have contributed two $(1+\epsilon)$-approximation algorithms for computing the minimum-volume bounding box problem for point sets in $\mathbb{R}^3$, both with nearly linear complexity. The running times of their algorithms are $O(n + 1/\epsilon^{4.5})$ and $O(n \log n + n/\epsilon^3)$.

Numerous heuristics have been proposed for computing a box which encloses a given set of points. The simplest heuristic is naturally to compute the axis-aligned bounding box of the point set. Two-dimensional variants of this heuristic include the well-known $R-tree$, the *packed* $R-tree$ [7], the $R^*-tree$ [8], the $R^+-tree$ [3], etc. A frequently used heuristic for computing a bounding box of a set of points is based on *principal component analysis*. The principal components of the point set define the axes of the bounding box, and the dimension of the bounding box along an axis is given by the extreme values of the projection of the points on the corresponding axis. Two distinguished applications of this heuristic are OBB-tree [4] and BOXTREE [1], hierarchical bounding box structures, which support efficient collision detection and ray tracing. Computing a bounding box of a set of points in $\mathbb{R}^2$ and $\mathbb{R}^3$ by PCA is quite fast, it requires linear time. To avoid the influence of the distribution of the point set on the directions of the PCs, a possible approach is to consider only the boundary of the convex hull of the point set. Thus, the complexity of the algorithm increases to $O(n \log n)$. The popularity of this heuristic besides its speed, lies in its easy implementation and in the fact that usually, PCA bounding boxes are tight-fitting.

We are not aware of any previous published results about the quality of the bounding boxes obtained by PCA. Here we give guarantees on the approximation factor of bounding boxes of convex polygons in $\mathbb{R}^2$ and convex polyhedra in $\mathbb{R}^3$.

The paper is organized as follows. In section 2 we review the basics of principal component analysis. In Section 3 we give lower bounds on the approximation factor of PCA bounding boxes in $\mathbb{R}^2$ and $\mathbb{R}^3$, and in Section 4 an upper bound in $\mathbb{R}^2$. We conclude with future work and open problems in Section 5.

## 2 Principal Component Analysis

The central idea and motivation of PCA [5] (also known as the Karhunen-Loeve transform, or the Hotelling transform) is to reduce the dimensionality of a data set by identifying *the most significant directions (principal components)*. Let $X = \{x_1, x_2, \ldots, x_m\}$, where $x_i$ is a $d$-dimensional vector, and $c = (c_1, c_2, \ldots, c_d)$ be the center of gravity of $X$. For $1 \le k \le d$, we use $x_{ik}$ to denote the $k$th coordinate of the vector $x_i$. Given two vectors $u$ and $v$, we use $\langle u, v\rangle$ to denote their inner product. For any unit vector $v \in \mathbb{R}^d$, the *variance of $X$ in direction $v$* is

$$var(X, v) = \frac{1}{m}\sum_{i=1}^{m} \langle x_i - c, v\rangle^2. \qquad (1)$$

The most significant direction corresponds to the unit vector $v_1$ such that $var(X, v_1)$ is maximum. In general, after identifying the $j$ most significant directions $B_j = \{v_1, v_2, \ldots, v_j\}$, the $(j+1)$th most significant direction corresponds to the unit vector $v_{j+1}$ such that $var(X, v_{j+1})$ is maximum among all unit vectors perpendicular to $v_1, v_2, \ldots, v_j$.

[*]Institut für Informatik, Freie Universität Berlin, Germany, {darko, knauer, kriegel, rote}@inf.fu-berlin.de

It can be verified that for any unit vector $v \in \mathbb{R}^d$,

$$var(X, v) = \langle Cv, v\rangle, \qquad (2)$$

where $C$ is the *covariance matrix* of $X$. $C$ is a symmetric $d \times d$ matrix where the $ij$-th component, $C_{ij}, 1 \le i, j \le d$, is defined as

$$C_{ij} = \frac{1}{m}\sum_{k=1}^{m}(x_{ik} - c_i)(x_{jk} - c_j). \qquad (3)$$

The procedure of finding the most significant directions, in the sense mentioned above, can be formulated as an eigenvalue problem. Namely, it can be shown that, if $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_d$ are the eigenvalues of $C$, then the unit eigenvector $v_j$ for $\lambda_j$ is the $j$th most significant direction. It follows that all $\lambda_j$s are non-negative as $\lambda_j = var(X, v_j)$. Since the matrix $C$ is symmetric positive definite, its eigenvectors are orthogonal. The following result summarizes the above background knowledge on PCA. For any set $S$ of orthogonal unit vectors in $\mathbb{R}^d$, we use $var(X, S)$ to denote $\sum_{v \in S} var(X, v)$.

**Lemma 1** *For $1 \le j \le d$, let $\lambda_j$ be the j-th largest eigenvalue of $C$ and let $v_j$ denote the unit eigenvector for $\lambda_j$. Let $B_j = \{v_1, v_2, \ldots, v_j\}$, $sp(B_j)$ be the linear subspace spanned by $B_j$, and $sp(B_j)^\perp$ be the orthogonal complement of $sp(B_j)$. Then $\lambda_1 = \max\{var(X, v)$ : unit vector $v$ in $\mathbb{R}^d\}$ and for any $2 \le j \le d$,*

*i) $\lambda_j = \max\{var(X, v)$ : unit vector $v$ in $sp(B_{j-1})^\perp\}$.*

*ii) $\lambda_j = \min\{var(X, v)$ : unit vector $v$ in $sp(B_j)\}$.*

*iii) $var(X, B_j) \ge var(X, S)$ for any set $S$ of $j$ orthogonal unit vectors.*

Since the covariance matrix depends on the distribution of the points, there is not necessarily a strong correlation between the eigenvectors and the directions of the axes of the minimal bounding box. Consider for example the situation when a significant number of points is located in a small part of the space, see figure 1. Moreover, by adding points inside or on the boundary of the convex hull of the point set, the PCA bounding box can arbitrarily vary between the minimum-volume bounding box and maximum-volume bounding box of the convex hull of the point set. To overcome this problem, one possible approach is to consider only the points on the boundary of the convex hull of the point set when the covariance matrix is computed. This is the approach we take in the rest of the paper. This leads us to so-called continuous PCA. In that case, $X$ is a continuous set of $d$-dimensional vectors and it can be verified that the derivations and the lemma above also hold.

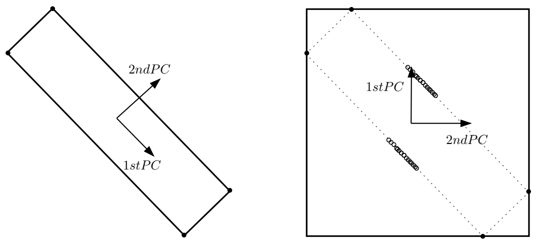


Figure 1: Four points and its PCA bounding-box (left). Dense collection of additional points significantly affect the orientation of the PCA bounding-box (right).

## 3 Lower Bounds

The following connection between hyperplane reflective symmetry and principal components will help us to derive the lower bounds of the approximation factor of the PCA bounding boxes.

**Theorem 2** *Let $P$ be a $d$-dimensional point set symmetric with respect to a hyperplane $H$. Then, a principal component of $P$ is orthogonal to $H$.*

**Proof.** Without loss of generality, we can assume that the hyperplane of symmetry is spanned by the last $n-1$ standard base vectors of the $d$-dimensional space and the center of gravity of the point set coincides with the origin of the $d$-dimensional space, i.e., $c = (0, 0, \ldots, 0)$. Then, the components $C_{1j}$ and $C_{j1}$, for $2 \le j \le d$, are 0 and the covariance matrix has the form:

$$C = \begin{bmatrix} C_{11} & 0 & \ldots & 0 \\ 0 & C_{22} & \ldots & C_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & C_{d2} & \ldots & C_{dd} \end{bmatrix} \qquad (4)$$

Its characteristic polynomial has the form:

$$det(C - \lambda\, I) = (C_{11} - \lambda)f(\lambda) \qquad (5)$$

where $f(\lambda)$ is a polynomial of degree $d-1$, with coefficients determined by the elements of the $(d-1) \times (d-1)$ submatrix of $C$. From this it follows that $C_{11}$ is a solution of the characteristic equation, i.e., it is an eigenvalue of $C$ and the vector (1, 0, ...,0) is its corresponding eigenvector (principal component), which is orthogonal to the assumed hyperplane of symmetry. $\square$

### 3.1 $\mathbb{R}^2$

We obtain a lower bound in $\mathbb{R}^2$ from a rhomb. Let its side length be $a$. Since the rhomb is symmetric, its PCs coincide with its diagonals. On the left side in figure 2 its optimal-area bounding boxes, for 2 different angles, are shown, and on the right side its corresponding PCA bounding boxes. As the rhomb's angles approach 90°, its optimal-area bounding box approaches a square with side length $a$, and the PCA bounding box a square with side length $\sqrt{2}a$. So, the ratio between the area of the PCA bounding box and the area of the optimal-area bounding box in the limit goes to 2.

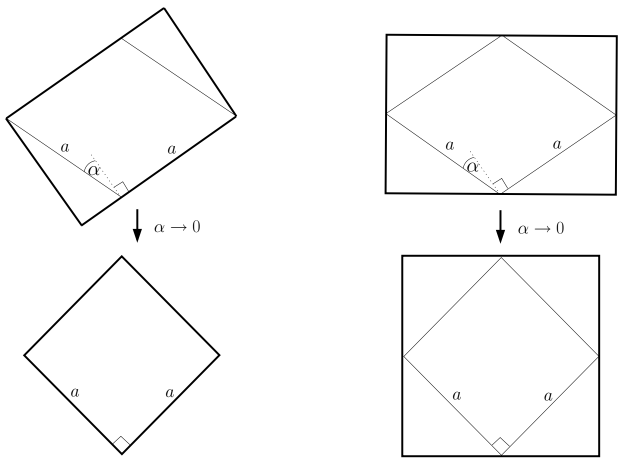


Figure 2: An example which gives us the lower bound of the area of the PCA bounding box of an arbitrary convex polygon in $\mathbb{R}^2$.

**Proposition 3** *In general, the ratio between the area of the PCA bounding box and optimal-area bounding box of a convex polygon cannot be smaller than 2.*

### 3.2 $\mathbb{R}^3$

We obtain a lower bound in $\mathbb{R}^3$ from a square dipyramid, having a rhomb with side length $\sqrt{2}$ as a base. Its other side lengths are $\frac{\sqrt{3}}{2}$. Similarly as in $\mathbb{R}^2$, we consider the case when its base, the rhomb, in limit approaches the square. Then the ratio of the volume of the bounding box on the left side in figure 3, and the volume of its PCA bounding box, on the right in figure 3, goes to 4.

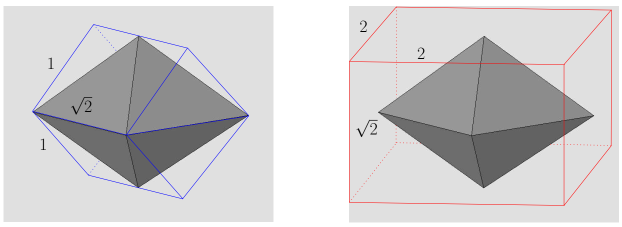


Figure 3: An example which gives the lower bound of the volume of the PCA bounding box of an arbitrary convex polygon in $\mathbb{R}^3$.

**Proposition 4** *In general, the ratio between the volume of the PCA bounding box and optimal-volume bounding box of a convex polyhedron cannot be smaller than 4.*

## 4 Upper Bound in $\mathbb{R}^2$

Let $P$ be a set of points in $\mathbb{R}^2$, and $\mathcal{P}$ the boundary of its convex hull. $\mathcal{P}$, its PCA bounding box and the line $l_{pca}$, which coincides with the 1st PC of $\mathcal{P}$, are given in the left part of figure 4. The optimal bounding box and the line $l_{\frac{1}{2}}$, going through the middle of its smaller side, parallel with its longer side, are given in the right part of figure 4.

The sides of any bounding box of $P$, $BB(P)$ (let us denote them with $a$ and $b$, s.t. $a \geq b$) cannot be larger than the diameter of $P$. From the other side, it is true that $diam(P) \leq diam(BB(P)) \leq \sqrt{2}a$. So we have the following relation

$$a_{pca} \leq diam(P) \leq \sqrt{2}a_{opt}. \qquad (6)$$

We denote with $d^2(\mathcal{P}, l)$ the integral of the squared

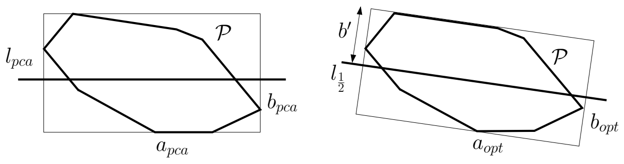


Figure 4: PCA bounding-box and the optimal bounding-box of the polygon $\mathcal{P}$.

distances of the points of $\mathcal{P}$ to the arbitrary line $l$, i.e. $d^2(\mathcal{P}, l) = \int_{x\in\mathcal{P}} d^2(x, l)ds$. From continuous version of lemma 1, part ii), follows that $l_{pca}$ is the best fitting line in the sense that it minimize the sum of squared distances, and therefore

$$d^2(\mathcal{P}, l_{pca}) \leq d^2(\mathcal{P}, l_{\frac{1}{2}}). \qquad (7)$$

We denote with $\mathcal{BB}_{\mathcal{OPT}}$ the boundary of the optimal bounding box of the $\mathcal{P}$. It is true that

$$\begin{aligned} d^2(\mathcal{P}, l_{\frac{1}{2}}) &\leq d^2(\mathcal{BB}_{\mathcal{OPT}}, l_{\frac{1}{2}}) \\ &= \frac{{b_{opt}}^2 a_{opt}}{2} + \frac{{b_{opt}}^3}{6}. \end{aligned} \qquad (8)$$

Due to space limitation, we leave the proof of (8) to a full paper. Now we look at $\mathcal{P}$ and its PCA bounding

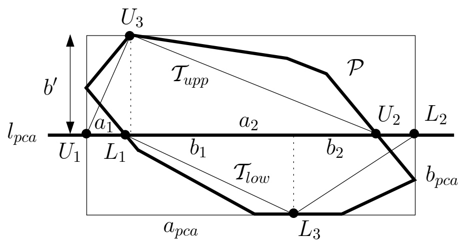


Figure 5: Lower bound for $d^2(\mathcal{P}, l_{pca})$.

box (figure 5). $l_{pca}$ divides $\mathcal{P}$ into an upper and a lower part, $\mathcal{P}_{upp}$ and $\mathcal{P}_{low}$. Let us denote with $l_{upp}$ the orthogonal projection of $\mathcal{P}_{upp}$ onto $l_{pca}$, with $U_1$ and $U_2$ as its extreme points, and with $l_{low}$ the orthogonal projection of $\mathcal{P}_{low}$ onto $l_{pca}$, with $L_1$ and $L_2$ as its extreme points. Since $\mathcal{P}$ is convex, the following relations hold:

$$|l_{upp}| \geq \frac{b'}{b_{pca}} a_{pca}, \text{ and } |l_{low}| \geq \frac{b_{pca} - b'}{b_{pca}} a_{pca}. \quad (9)$$

We inscribe in $\mathcal{P}_{upp}$ a triangle $\mathcal{T}_{upp}(\triangle U_1 U_2 U_3)$, and in $\mathcal{P}_{low}$ a triangle $\mathcal{T}_{low}(\triangle L_1 L_2 L_3)$. It then holds that

$$d^2(\mathcal{P}, l_{pca}) = d^2(\mathcal{P}_{upp} \bigcup \mathcal{P}_{low}, l_{pca}) \geq d^2(\mathcal{T}_{upp} \bigcup \mathcal{T}_{low}, l_{pca}) = d^2(\mathcal{T}_{upp}, l_{pca}) + d^2(\mathcal{T}_{low}, l_{pca}). \quad (10)$$

Due to space limitation, we leave also the proof of (10) to a full paper. The value

$$d^2(\mathcal{T}_{upp}, l_{pca}) = \frac{b'^2}{3}\left(\sqrt{a_1^2 + b'^2} + \sqrt{a_2^2 + b'^2}\right)$$

is minimal when $a_1 = a_2 = \frac{|l_{upp}|}{2}$. So with (9) we get

$$d^2(\mathcal{T}_{upp}, l_{pca}) \geq \frac{b'^3}{3b_{pca}}\left(\sqrt{a_{pca}^2 + 4b_{pca}^2}\right).$$

Analogously, we have for the lower part:

$$d^2(\mathcal{T}_{low}, l_{pca}) \geq \frac{(b_{pca} - b')^3}{3b_{pca}}\left(\sqrt{a_{pca}^2 + 4b_{pca}^2}\right).$$

The sum $d^2(\mathcal{T}_{upp}, l_{pca}) + d^2(\mathcal{T}_{low}, l_{pca})$ is minimal when $b' = \frac{b_{pca}}{2}$. This, together with (10), gives:

$$d^2(\mathcal{P}, l_{pca}) \geq \frac{b_{pca}^2}{12}\sqrt{a_{pca}^2 + 4b_{pca}^2}. \quad (11)$$

Combining (7), (8) and (11) we have:

$$\frac{1}{2} a_{opt} b_{opt}^2 + \frac{1}{6} b_{opt}^3 \geq \frac{b_{pca}^2}{12}\sqrt{a_{pca}^2 + 4b_{pca}^2}. \quad (12)$$

Let $a_{pca} = \alpha a_{opt}$ and $b_{pca} = \beta b_{opt}$. Replacing $b_{pca}^2$ with $\beta^2 b_{opt}^2$ in (12), we obtain:

$$6a_{opt} + 2b_{opt} \geq \beta^2 \sqrt{a_{pca}^2 + 4b_{pca}^2} \geq \beta^2 a_{pca}.$$

Replacing further $a_{pca}$ with $\alpha a_{opt}$, we obtain:

$$6a_{opt} + 2b_{opt} \geq \beta^2 \alpha a_{opt} = \frac{\beta^2 \alpha}{8}(6a_{opt} + 2a_{opt}).$$

Since $a_{opt} \geq b_{opt}$, we have

$$6a_{opt} + 2b_{opt} \geq \frac{\beta^2 \alpha}{8}(6a_{opt} + 2b_{opt}),$$

and from this

$$\beta \leq \sqrt{\frac{8}{\alpha}}. \quad (13)$$

Finally, from (6) and (13) we obtain:

$$\begin{aligned} \frac{area(BB_{PCA}(\mathcal{P}))}{area(BB_{OPT}(\mathcal{P}))} &= \frac{a_{pca} b_{pca}}{a_{opt} b_{opt}} = \alpha\beta \leq \sqrt{8\sqrt{\alpha}} \\ &\leq \sqrt{8\sqrt{2}} \approx 3.3635856. \end{aligned}$$

We summarize this result in the following theorem:

**Theorem 5** *Let $\mathcal{P}$ be the boundary of the convex hull of the point set $P \subset \mathbb{R}^2$. The ratio between the area of the PCA bounding box of $\mathcal{P}$ and the area of its optimal bounding box is bounded from above by 3.3636.*

## 5 Future Work and Open Problems

Improving the upper bound in $\mathbb{R}^2$, as well as obtaining an upper bound in $\mathbb{R}^3$ are our current interests. A variant of the PCA bounding box problem, where instead of considering only the points on the boundary of the convex hull all points from the convex hull are taken into account, is also of interest. A very demanding open problem is to get an approximation factor of PCA bounding boxes in arbitrary dimension.

## References

[1] G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell and A. Tal. BOXTREE: A Hierarchical Representation for Surfaces in 3D. *Computer Graphics Forum*, 1996, vol. 15, no.3, pages 387–396.

[2] G. Barequet and S. Har-Peled. Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in 3D. *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, 1999, pages 82–91.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *In ACM SIGMOD Int. Conf. on Manag. of Data*, 1990, pages 322–331.

[4] S. Gottschalk, M. C. Lin and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Proc. SIGGRAPH 1996*, pages 171–180.

[5] I. Jolliffe. Principal Component Analysis. *Springer-Verlag, New York, 2nd ed.*, 2002.

[6] J. O'Rourke. Finding Minimal Enclosing Boxes. *Int. J. Comp. Info. Sci. 14 (1985)*, pages 183–199.

[7] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-Trees. *In Proc. of the ACM SIGMOD*, 1985, pages 17–31.

[8] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multidimensional objects. *In Proc. 13th VLDB Conference*, 1987, pages 507–518.

[9] G. Toussaint. Solving geometric problems with the rotating calipers. *Proc. IEEE MELECON'83*, May 1983.

# Algorithms for Maximizing the Volume of Intersection of Polytopes

Komei Fukuda* Takeaki Uno[1]†

**Abstract:** In this paper, we address the problem of maximizing the volume of the intersection of polytopes in $\mathbb{R}^d$ by translation. We show that (1) the $d$th root of the objective function is concave, thus the problem can be solved oracle polynomial time by ellipsoidal method, and (2) the problem can be solved in strongly polynomial time in dimension two even for non-convex polygons.

## 1 Introduction

Let $A$ and $B$ be two polygons in Euclidean plane where the position of $A$ is fixed and the body $B$ can be translated freely. Then, the intersection of $A$ and $B$ changes as the move. Here we consider the *intersection maximization problem*, that is, to maximize the volume of the intersection by translation. This problem is easy to state, but it appears that it has not been investigated in depth. In fact, this problem has been recently proposed as an open problem at an Oberwolfach workshop by P. Brass [2]. Let us consider a simple example in Figure 1. By shifting horizontally the triangle from left to right, the volume of the intersection of the triangle and the square initially increases as a convex function in the translation variable. Once the right half of the triangle is contained in the square, the volume increase as a concave function. Thus, the volume of the intersection has both features of convex and concave functions. One can also see that the function is continuous, but not differentiable. Consequently, maximization of such functions may not be done in a straightforward manner.

In this paper, we address this problem from the computational point of view. Firstly, we show that the $d$th root of the objective function is concave for any finite number of $d$-dimensional convex polytopes. It follows that we can solve the problem in oracle polynomial time where the oracle is the computation of the volume and its gradient. We also show that the hyperplanes spanned by the facets of the polytopes define an arrangement where the objective function is a simple polynomial function in each of its regions. By using these properties, we propose a strongly polynomial time algorithm for the case of two convex polygons in the plane. Its time complexity is $O(n^2 \log^2 n)$ and its space complexity is $O(n)$ if they both have at most $n$ edges. We further propose an enumeration based algorithm for the problem with two non-convex polygons which runs in $O(n^4)$ time and $O(n^2)$ space.

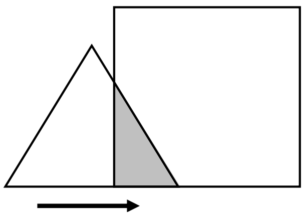


Figure 1: The volume of the intersection of two polygons

## 2 Preliminaries

We denote the $d$ dimensional Euclidean space by $\mathbb{R}^d$. For a convex body $P$ in $\mathbb{R}^d$ (i.e. a compact convex set with nonempty interior), we denote its volume by $vol(P)$. For a vector $h$ in $\mathbb{R}^d$, $P + h$ is the convex body obtained by translating $P$ by $h$, i.e., $P + h = \{x + h | x \in P\}$. Whenever there is no confusion, we call a face of $B+h$ face of $B$. For a family $\mathcal{P}$ of convex bodies $P_1, \ldots, P_k$, we denote its intersection by $\cap(\mathcal{P})$, i.e., $\cap(\mathcal{P}) = \cap_{i=1}^k P_i$.

For a given set of polytopes $\mathcal{P} = \{P_1, \ldots, P_k\}$ in $\mathbb{R}^d$, the intersection maximization problem is to maximize $vol(\cap(\mathcal{P} = \{P_1, P_2 + h_2, \ldots, P_k + h_k\}))$ subject to $h_i \in \mathbb{R}^d$ for $i = 2, \ldots, k$.

## 3 Convexity on the Intersection Volume of Polytopes

Our first result is the following theorem.

**Theorem 1** *For any two convex bodies $A$ and $B$ in $\mathbb{R}^d$, the function $(vol(\cap(\{A, B+h\})))^{1/d}$ is concave in $h$ over the region of nonempty intersection, $\Omega = \{h \in \mathbb{R}^d | \cap (\{A, B + h\}) \neq \emptyset\}$.*

To prove the theorem, we use the Brunn-Minkowski theorem below. Let us denote by $H(z, d)$ the hyperplane in $\mathbb{R}^d$ given by $x_d = z$.

**Theorem 2 (Brunn-Minkowski, see [1])** *For any convex body $P$ in $\mathbb{R}^d$, the function $(vol(\cap(\{P, H(z, d)\})))^{1/(d-1)}$ is concave in $z$ over the region $\Omega = \{z \in \mathbb{R}^d | \cap (\{P, H(z, d)) \neq \emptyset\}$.*

*Institute for Operations Research and Institute of Theoretical Computer Science, ETH Zentrum, CH-8092 Zurich, and IMA/ROSO, EPF Lausanne, CH-1015 Lausanne, Switzerland, e-mail:fukuda@ifor.math.ethz.ch

†National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan, e-mail:uno@nii.jp

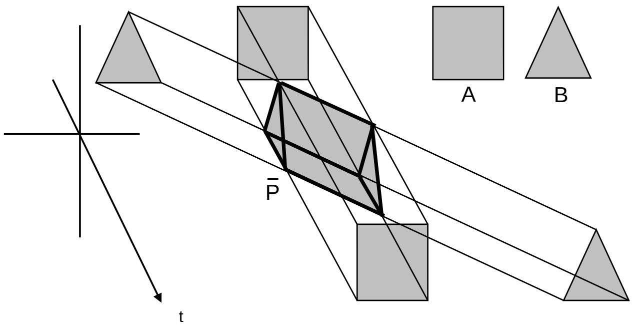


Figure 2: Polytopes $\bar{A}$, $\bar{B}$, and $\bar{P}$

*Proof:* (of Theorem 1) Let $h, h' \in \mathbb{R}^d$ be any vectors such that both $\cap(\{A, B+h\})$ and $\cap(\{A, B+h'\})$ are non-empty. It suffices to show that the function is concave over the line segment connecting $h$ and $h'$. Let

$$\begin{aligned}\bar{A} &= \{(x_1,\dots,x_d,\lambda) \mid 0 \le \lambda \le 1, (x_1,\dots,x_d) \in A\},\\ \bar{B} &= \{(x_1,\dots,x_d,\lambda) \mid 0 \le \lambda \le 1,\\ &\qquad (x_1,\dots,x_d) \in B + (\lambda h + (1-\lambda)h'))\}.\end{aligned}$$

Since both sets are convex, $\bar{P} = \bar{A} \cap \bar{B}$ is also convex, see Figure 2. Consequently, the intersection of $H(\lambda, d+1)$ and $\bar{P}$ is

$$\begin{aligned}&H(\lambda, d+1) \cap \bar{P}\\ &= \{(x_1,\dots,x_d,\lambda) \mid (x_1,\dots,x_d) \in A,\\ &\qquad (x_1,\dots,x_d) \in B + (\lambda h + (1-\lambda)h')\},\\ &= \{(x_1,\dots,x_d,\lambda)\\ &\qquad (x_1,\dots,x_d) \in A \cap B + (\lambda h + (1-\lambda)h')\}.\end{aligned}$$

Thus, the intersection is a lifted copy of $\cap(\{A, B + (\lambda h + (1-\lambda)h')\})$. The theorem then follows directly from the Brunn-Minkowski Theorem. ▮

The theorem above can be easily extended to the intersection of several polytopes.

**Theorem 3** *For any convex bodies $P_1, \dots P_m$ in $\mathbb{R}^d$, the function $(vol(\cap(P_1+h_1, P_2+h_2, \dots P_m+h_m)))^{1/d}$ is concave in $h = (h_1, \dots, h_m)$ over $\Omega = \{h \mid \cap(P_1 + h_1, P_2 + h_2, \dots P_m + h_m)) \neq \emptyset\}$.*

It follows from Theorem 1 that any locally maximum solution is a global maximum solution. This also implies that the volume of the intersection is a unimodal (quasiconcave) function. Moreover, it is semistrictly quasiconcave. A function $f(x)$ is called *semistrictly quasiconcave* if $f(y) < f(\lambda x + (1-\lambda)y)$ holds for any $x$ and $y$ with $f(x) > f(y)$, and $0 < \lambda < 1$.

For a given point $x$ and a function $f$ having a maximum solution, a hyperplane is called separating hyperplane if it separates $x$ from the set of maximum solutions. Any function can be maximized by ellipsoidal method with finding polynomially many separating hyperplanes. A separating hyperplane of a non-differentiable concave function can be obtained from its subgradient, thus we obtain the following theorem.

**Theorem 4** *The problem of maximizing the intersection of d-dimensional polytopes by translation without rotation can be solved in oracle polynomial time in the input size, where the oracle is to compute the volume and its subgradient of the intersection.*

In the next section, we analyze the structure of the objective function so that we can construct a combinatorial algorithm that terminates in strongly polynomial time when $d = m = 2$.

## 4 Decomposing the Domain into Regions of Equivalence

Let $A$ and $B$ be two convex polygons in $\mathbb{R}^2$, and $h$ be a vector in $\mathbb{R}^2$. Here we regard $h$ as variables. For a vertex $v$ of $\cap(\{A, B+h\})$, we define its *topological representation* by the set of facets (edges) of $A$ and $B$ containing $v$. The position of $v$ is given by the intersection of these facets, and thus we can represent the position of $v$ as a unique solution to the linear system given by the facets. The solution is a linear function of $h$. We call this the *functional representation* of $v$.

Suppose that we are given functional representations of the vertices of $\cap(\{A, B+h\})$, and consider its volume. In general, the volume of a polytope in $\mathbb{R}^d$ is the sum of the volumes of simplexes in a triangulation of the polytope. The volume of each simplex is obtained from the determinant of a matrix consisting of its vertices properly lifted. Therefore, the volume is a polynomial of degree at most $d$. In the plane, the volume is a quadratic function in $h$. We call this function the *volume function*.

The following lemma is important.

**Lemma 5** *The volume function of $\cap(\{A, B+h\})$ in $h$ and that of $\cap(\{A, B+h'\})$ in $h'$ are identical if the topological representations of the vertices are the same.*

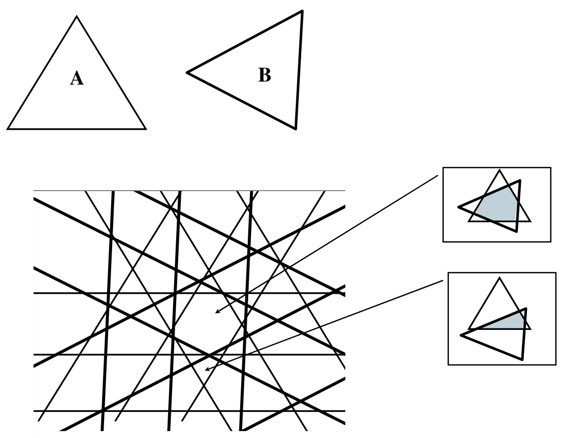


Figure 3: Arrangements by the hyperplanes induced by the facets of $A$ and $B$

The translations $h$ that induce the same topological representations form an equivalence class. Each equivalence class is determined by a set of linear inequalities which represent conditions such as "a vertex of $A$ (resp., $B+h$) is in a half space induced by a facet of $B+h$ (resp., $A$)." We denote the set of the hyperplanes (lines) being the boundary of an equivalence class by $\mathcal{H}(\{A, B\})$. The cardinality of $\mathcal{H}(\{A, B\})$ is $O(n^2)$.

Consider the arrangement by the hyperplanes in $\mathcal{H}(\{A, B\})$ (see Figure 3). Then, in a region (i.e a full-dimensional cell) of the arrangement, the set of the topological representations of the vertices does not change. Thus, in each equivalence region, the intersection maximization problem is just a non-convex non-concave quadratic programming with two variables. It can be solved by evaluating the values of the objective function on the vertices, the edges, and the points satisfying that the gradient is 0. It can be done in linear time in the number of edges of the region. Enumerating all regions in the arrangement by $O(n^2)$ lines takes $O(n^4)$ time[3]. The volume function of a region can be computed in constant time from the volume function of a region adjacent to it. Consequently, we can solve the intersection maximizing problem in $O(n^4)$ time and $O(n^2)$ space. Furthermore, this method does not depend on the convexity of two polygons. Therefore, we obtain the following theorem.

**Theorem 6** *The problem of maximizing the volume of the intersection of two non-convex polygons in the plane can be solved in $O(n^4)$ time and $O(n^2)$ space.*

Note that one can extend the theorem to general $d$ dimension with $k$ non-convex polytopes. However, the problem is no longer easy, since the volume function is a polynomial of higher degrees, and the number of the regions is huge, exponential in both $k$ and $d$.

## 5 Binary Search Algorithm

We here assume that the objective function is perturbed so that the optimal solution is unique. Suppose that $H$ is a hyperplane in $\mathcal{H}(\{A, B\})$, and $h$ is the point in $H$ maximizing $vol(\cap(\{A, B+h\}))$. Then, by looking at the gradient of $vol(\cap(\{A, B+h\}))$, we can check which half space induced by $H$ contains an optimal solution. Using this fact, one can construct a binary search algorithm.

Every hyperplane in $\mathcal{H}(\{A, B\})$ is a translation of a hyperplane spanned by a facet of $A$ or $B$. It follows that the hyperplanes in $\mathcal{H}(\{A, B\})$ can be partitioned into groups $\mathcal{H}_1, \ldots, \mathcal{H}_{2n}$ such that each group is composed of hyperplanes parallel to each other. Suppose that $\mathcal{H}_i = \{H_1, \ldots, H_k\}$ are sorted in their direction. Then, there is $H_j$ such that an optimal solution is in the area between $H_j$ and $H_{j+1}$. We call the area between $H_j$ and $H_{j+1}$ the *optimal slab* of $\mathcal{H}_i$. The intersection of the optimal slabs of all $\mathcal{H}_i$ gives the region in which an optimal solution is contained. The optimal slab of $\mathcal{H}_i$ can be found by binary search on $\mathcal{H}_i$, by solving $O(\log n)$ intersection maximization problems on a hyperplane.

Although a straightforward binary search on a line needs $O(n^2)$ preprocessing time and $O(n^2)$ memory, one can reduce them to $O(n \log n)$ time and $O(n)$ space by a median finding like algorithm and implicit representation of $\mathcal{H}_i$.

## 6 Solving the Problem on a Line

For a hyperplane (line) $H \in \mathcal{H}_i$, we present a method to find an optimal solution $h^*$ in $H$, which maximizes the volume function restricted to $H$. Since a line is partitioned into intervals by hyperplanes in $\mathcal{H}(\{A, B\})$, our objective is to find the *optimal interval*, i.e., the interval containing the optimal solution.

Let $x_1, \ldots, x_m$ be the intersection points of $H_i$ and hyperplanes in $\mathcal{H}(\{A, B\})$ which are not parallel to

$H$. If $x_1, \ldots, x_m$ are sorted in the order of their positions on $H$, we can easily perform binary search in $O(n \log n)$ time. However, to compute all $x_1, \ldots, x_m$, we need $O(n^2)$ time. Moreover, we need $O(n^2)$ space to keep $\mathcal{H}(\{A, B\})$ and $x_1, \ldots, x_m$ in memory. Thus, here we consider how to execute binary search without computing $x_1, \ldots, x_m$.

The basic idea of the binary search is as follows. Suppose that the groups of hyperplanes are $\mathcal{H}_1, \ldots, \mathcal{H}_p$, and the hyperplanes in each $\mathcal{H}_i$ are sorted. Here we do not include the hyperplanes parallel to $H$ in them. We denote by $H_i^*$ the median of $\mathcal{H}_i$ in the sorted order, and the intersection point of $H$ and $H_i^*$ by $x_i^*$. Then, we find $H_z^*$ such that $x_z^*$ is "middle" of $x_1^*, \ldots, x_p^*$. Here the middle means that $z$ satisfies

$$\sum_{i=1}^{z} |\mathcal{H}_i| \geq (\sum_{i}^{p} |\mathcal{H}_i|)/2, \text{and}$$
$$\sum_{i=z}^{p} |\mathcal{H}_i| \geq (\sum_{i}^{p} |\mathcal{H}_i|)/2.$$

Note that $\sum_i^p |\mathcal{H}_i|$ is the number of intersection points generated by $H$ and hyperplanes in $\mathcal{H}_1, \ldots, \mathcal{H}_p$. Thus, by looking at the gradient at $x^*$, we can determine at least 1/4 of the hyperplanes which do not give the endpoints of the optimal interval. More precisely, for each $\mathcal{H}_i = \{H_1, \ldots, H_l\}$, suppose that $x_i^*$ is on the side without the optimal solution. The both endpoints of the optimal interval are in the other side, thereby not given by $H_1, \ldots, H_i^*$ (or $H_i^*, \ldots, H_l$). Therefore, we can reduce such $\mathcal{H}_i$ to $\mathcal{H}_i \setminus \{H_1, \ldots, H_i^*\}$. By repeating this process, $\sum_i^p |\mathcal{H}_i|$ decreases by a constant factor each time. This yields a binary search with $O(\log n)$ steps.

Finding $x_z^*$ can be done in $O(n)$ time by a median finding like method. We find the median, usual meaning of median, of $x_1, \ldots, x_p$ by a median finding algorithm in $O(n)$ time. Then we can see that at least a half of $x_1, \ldots, x_p$ will not be $x_z^*$. In this way, we can iteratively reduce the candidates, and find $x_z^*$ in $O(n)$ time.

The last remaining problem is to reduce $O(n^2)$ space to store each sorted $\mathcal{H}_i$ in memory. For this, we divide $\mathcal{H}_i$ into two groups, and keep them in memory implicitly with $O(1)$ space. Let $v_1, \ldots, v_n$ be the vertices of $A$ sorted in the clockwise order. For an edge $e$ in $B$, we denote one of its orthogonal direction by $D(e)$. Let $v^*$ and $v^{**}$ be vertices which maximizes and minimizes $D(e)$, respectively. We divide $v_1, \ldots, v_n$ into two groups such that one group is composed of vertices from $v^*$ to $v^{**}$, and the other is composed of vertices from $v^{**}$ to $v^*$. We consider that $v_1, \ldots, v_n$ is a cyclic sequence. Then, we can see that the order of the hyperplanes induced by $e$ and vertices from $v^*$ to $v^{**}$ is a sorted order, and that by vertices from $v^{**}$ to $v^*$ is also a sorted order. Thus, keeping these by two end vertices, we can implicitly keep them in memory with $O(1)$ space.

Now we describe the whole algorithm.

**Algorithm** MaxIntersection ($A$ ,$B$)
```
1. sort the vertices of $A$ and $B$ in the clockwise
    order, respectively
2. for each edge $e$ in $A$ and $B$
3.   compute $v^*$ and $v^{**}$
4.   make implicit representation of groups,
       $e$ and vertices from $v^*$ to $v^{**}$, and
       $e$ and vertices from $v^{**}$ to $v^*$
5. endfor
6. for each group $\mathcal{H}_i = \{H_1, \ldots, H_l\}$
    //binary search for the optimal slab
7.   $p$ :=1, $q := k$
8.   while $p + 1 < q$
9.     find the optimal solution $h^*$ in $H_{(p+q)/2}$ by
         binary search
10.    set $p := (p + q)/2$ or set $q := (p + q)/2$
         according to the gradient at $h^*$
11.  endwhile
12. endfor
13. output the optimal solution in the intersection of
     optimal slabs
```

Finally, we have the following theorem.

**Theorem 7** *The maximization problem of the volume of two convex polygons can be solved in $O(n^2 \log^2 n)$ time and $O(n)$ space.* ▮

## Acknowledgment

We thank Professor Masakazu Kojima, Akiko Takeda of Tokyo Institute of Technology, Akiyoshi Shioura of Tohoku University, and Junya Goto of Tsukuba University for their answering out questions. This research is partially supported by Swiss National Science Foundation and Japan Society for the Promotion of Science.

## References

[1] K. Ball, "An Elementary Introduction to Modern Convex Geometry," *Flavors of Geometry, Math. Sci. Res. Inst. Publ.*, **31**, Cambridge Univ. Press, Cambridge, 1997.

[2] P. Brass, "A Lower Bound for Lebesgue's Universal Cover Problem," *Workshop of Discrete Geometry*, Mathematisches Forschungsinstitut Oberwolfach, 2005.

[3] H. Edelsbrunner and L. J. Guibas, "Topologically Sweeping an Arrangement," *J. Comput., Syst. Sci.*, **38**, 165-194, 1989.

[4] G. T. Toussaint, "A Simple Linear Algorithm for Intersecting Convex Polygons," *The Visual Computer* **1**, 118-123, 1985.

# From triangles to curves

Monique Teillaud

Projet Geometrica, INRIA Sophia Antipolis
BP 93, 06902 Sophia Antipolis Cedex, France
`Monique.Teillaud@sophia.inria.fr`

The objects studied in Computational Geometry were traditionally linear objects (points, line segments, triangles,...) and research on curved objects was quite theoretical.

Curves and surfaces have been considered from a more practical point of view for a few years, especially in Europe.[a] Cross-fertilization between researchers in Computational Geometry and Computer Algebra allowed advances that this talk will try to summarize.

On the implementation side, the talk will in particular mention the work in progress in the CGAL Open Source project.[b]

[a]Let us mention two European projects, ECG (Effective Computational Geometry for Curves and Surfaces - `http://www-sop.inria.fr/prisme/ECG/`) and ACS (Algorithms for Complex Shapes with certified topology and numerics - `http://acs.cs.rug.nl/`).

[b]`www.cgal.org`

# On Embedding a Graph on Two Sets of Points

Emilio Di Giacomo Giuseppe Liotta Francesco Trotta*

## 1 Introduction

Let $S_0, S_1, \ldots, S_{k-1}$ be $k$ sets of points such that the points of $S_i$ are colored with color $i$ ($i = 0, \ldots, k-1$). Let $G$ be a planar graph such that $|S_i|$ vertices of $G$ have color $i$, for every $0 \leq i \leq k-1$. A *$k$-chromatic point-set embedding* of $G$ on $S = S_0 \cup S_1 \cup \cdots \cup S_{k-1}$ is a crossing-free drawing of $G$ such that each vertex colored $i$ is mapped to a point of $S_i$, and each edge is a polygonal curve.

If $k = 1$ ("monochromatic" case), all the vertices and all the points have the same color, and therefore any vertex can be mapped on any point. In this case the algorithm that computes the drawing of $G$ on $S$ can choose the mapping between vertices and points as it is more convenient. Kauffman and Wiese [4] prove that every planar graph admits a monochromatic point-set embedding with at most two bends per edge on any given set of points.

If $k = n$, where $n$ is the number of vertices of $G$, each vertex of $G$ must be drawn on the unique point with the same color. Therefore the mapping between vertices and points is given as a part of the input and the drawing algorithm cannot change it. Pach and Wenger [5] prove that every planar graph has an $n$-chromatic point-set embedding on any given set of points such that each edge has $O(n)$ bends; they also prove that this bound is asymptotically optimal in the worst case even if $G$ is a path.

Given the two results above, a natural question arises: If two bends per edge are necessary and sufficient to solve the monochromatic point-set embedding problem [4] while $O(n)$ bends per edge are necessary and sufficient for the $n$-chromatic case [5], how many bends per edge do we need if the number of colors is a constant larger than one?

In this paper we continue the study, initiated in a previous work [1], of the apparently simple case of two colors. The two colors will be referred in the following as *red* and *blue* and the two set of points $S_0$ and $S_1$ will be denoted as $R$ and $B$. Notice that, by the result of Pach and Wenger [5], every planar graph has a bi-chromatic point-set embedding (2CPSE) with $O(n)$ bends per edge (arbitrarily map every red/blue vertex to a red/blue point and then use the drawing technique of [5]). Therefore the question that we ask is whether $O(n)$ bends per edge is also a lower bound for the 2CPSE problem and/or there are cases where a constant number of bends can be achieved. In [1] the simple family of bi-chromatic paths is considered and it is proved that every bi-chromatic path admits a 2CPSE on any two sets of red and blue points with at most one bend per edge.

The main results in this paper can be listed as follows. **(a)** In Section 3 we show that there exists a 2-colored tri-connected planar graph $G$ with $n \geq 8$ vertices and two sets of points $R$ and $B$ such that every 2CPSE of $G$ on $R \cup B$ has one edge that requires at least $\lceil \frac{n}{6} \rceil - 1$ bends. This proves that bi-chromatic point-set embeddability of a generic planar graph requires $O(n)$ bends per edge. Motivated by this result we investigate subclasses of planar graphs.

**(b)** In Section 4 we prove that every 2-colored caterpillar admits a 2CPSE on any two sets of red and blue points such that every edge of the drawing has at most two bends. We also prove that for properly 2-colored caterpillars (i. e. colored in such a way that no two adjacent vertices have the same color) the number of bends per edge can be reduced to one, which is worst-case optimal since not all bi-chromatic paths admit a 2CPSE with no bends per edge [3].

## 2 Preliminaries

Let $G = (V, E)$ be a planar graph. A *2-coloring* of $G$ is a partition of $V$ into two disjoint sets $V_b$ and $V_r$, the *blue vertices* and the *red vertices* respectively. A 2-coloring is *proper* if for every edge $(u, v) \in E$ we have $u \in V_b$ and $v \in V_r$. Let $R$ be a set of red points in the plane and let $B$ be a set of blue points in the plane. We say that $S = B \cup R$ is *equipollent with* $G$ if $|B| = |V_b|$ and $|R| = |V_r|$. A 2-colored planar graph $G$ is *bi-chromatic point-set embeddable* if it has a 2CPSE on *any* set of points equipollent with $G$. We denote by $c(x)$ the color of $x$, where $x$ can be a vertex, a point or a set of vertices/points with the same color.

Let $G$ be a planar graph. A *2-page book embedding* of $G$ is a crossing-free drawing of $G$ such that: (i) the vertices of $G$ are represented as points of a straight line called *spine*, and (ii) each edge is drawn as a simple Jordan curve completely contained in one of the two half-planes (*pages*) defined by the spine. A *subdivision* of a graph $G = (V, E)$ is a graph obtained from $G$ by replacing each edge $(u, v) \in E$ by a path with at least one edge whose endpoints are $u$ and $v$. Inter-

*Dipartimento di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia, {digiacomo, liotta, francesco.trotta}@diei.unipg.it nal vertices on this path are called *division* vertices. A 2*-page topological book embedding* of $G$ is a 2-page book embedding of a subdivision of $G$. If the number of division vertices for each edge is at most $d$ we say that $G$ has 2*-page topological book embedding with at most $d$ divisions.*

A *red-blue sequence* $\sigma$ is a set of collinear points such that each point is either red or blue. Let $G$ be a 2-colored planar graph and let $\sigma$ be a red-blue sequence equipollent with $G$. A 2-page (topological) book embedding of $G$ *consistent with* $\sigma$ is a 2-page (topological) book embedding of $G$ such that each vertex $v$ of $G$ is represented by a point $p$ of $\sigma$ and $c(v) = c(p)$. A 2-colored planar graph $G$ is 2*-page bi-chromatic (topological) book embeddable* if, for any red-blue sequence $\sigma$ equipollent with $G$, $G$ has a 2-page (topological) book embedding consistent with $\sigma$. If the number of division vertices for each edge is at most $d$, we say that $G$ is 2*-page bi-chromatic topological book embeddable with at most $d$ divisions.*

## 3 Curve Complexity of 2CPSEs

One can compute a drawing with $O(n)$ bends per edge by mapping every vertex $v \in G$ to a point $p \in S$ such that $c(v) = c(p)$ and then use the algorithm in [5]. Therefore every 2-colored planar graph is bi-chromatic point-set embeddable with $O(n)$ bends per edge. In this section we prove that such number of bends per edge can also be necessary for some configurations.

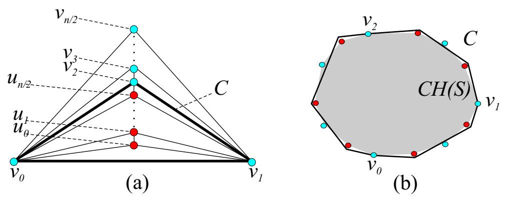


Figure 1: (a) The graph $G^*$ whose 2CPSE on the set of points (b) requires $\lceil \frac{n}{6} \rceil - 1$ bends per edge .

Let $G^*$ be one of the graph in the class of triconnected bi-chromatic planar graphs shown in Figure 1 (a) and let $S$ be a convex set equipollent with $G^*$ such that the red and blue points alternate along the shape of its convex hull. We can prove that in any 2CPSE of $G^*$ on $S$ one of the three edges of the cycle $C$ highlighted in Figure 1 has at least $\lceil \frac{n}{6} \rceil - 1$ bends (see [2]). The following theorem therefore holds.

**Theorem 1** *A 2-colored planar graph is bi-chromatic point-set embeddable with $O(n)$ bends per edge, which is worst-case optimal.*

Theorem 1 naturally raises the question about whether there exist families of 2-colored planar graphs that are bi-chromatic point-set embeddable with a constant number of bends. In next section we affirmatively aswer this question for caterpillars.

## 4 Computing 2CPSEs with at Most Two Bends per Edge

Bi-chromatic point-set embeddability can be studied by modelling the problem as a 2-page bi-chromatic topological book embeddability problem based on the following theorem, whose proof is omitted for brevity (see [2]).

**Theorem 2** *Let $G$ be a 2-colored planar graph. Then $G$ is bi-chromatic point-set embeddable with at most two bends per edge if and only if it is 2-page bi-chromatic topological book embeddable with at most one division.*

A *caterpillar* $G$ is graph that consists of a path, called the *body* of $G$, and of a (possibly empty) set of vertices adjacent to the body and having degree one. Based on Theorem 2, we describe a drawing algorithm that computes a 2-page bi-chromatic topological book embedding of $G$ consistent with $\sigma$ and with at most one division.

We need some additional notation. We denote as $v_0$, $v_1$, …, $v_h$ the vertices of the body of $G$. We denote as $G_0$ the graph consisting of vertex $v_0$ and as $G_k$ ($k = 1, \ldots, h$) the subgraph of $G$ induced by $v_0$, $v_1$, …, $v_{k-1}$ and by their adjacent vertices. We denote as $\mathcal{N}_k$ the set of vertices of $G_{k+1} \setminus G_k$, i.e. all vertices adjacent to $v_k$ in $G$ except $v_{k-1}$; also, we denote as $\mathcal{N}_k^-$ the set $\mathcal{N}_k \setminus \{v_{k+1}\}$, i.e. all the leaves of $G$ adjacent to $v_k$. We define $\mathcal{N}_{-1}$ as the graph consisting of vertex $v_0$.

We are going to describe a drawing algorithm, called `Cater-Draw` that is recursive with the number of vertices in the body of $G$. At Step $k$ of the recursion the subgraph induced by $\mathcal{N}_{k-1}$ is added to the current drawing. The output of the algorithm after $k$ steps is a drawing $\gamma_k$ that maintains a set of invariant properties. In the next sections we denote as $\sigma_k \subseteq \sigma$ the red-blue subsequence consisting of all points representing the vertices of $G_k$ in $\gamma_k$. The rightmost point of $\sigma_k$ is denoted as $\rho_k$. The set of all points of $\sigma \setminus \sigma_k$ that are to the left of $\rho_k$ in $\sigma$ is denoted as $NB_k$. The set of vertices that are after $\rho_k$ in $\sigma$ and whose color is $c$ ($c \in \{b, r\}$) is denoted as $F_k^c$.

Let $l$ be the line through the points of $\sigma_k$ and let $q$ be any point of $l$ ($q$ may or may not be an element of $\sigma$) and let $\pi$ be either the top or the bottom half-plane (page) defined by $l$. We say that $q$ is *accessible from $\pi$ in $\gamma_k$* if there is no edge $(u, v)$ in $G_k$ such that $q$ is between the point representing $u$ and the point representing $v$ in $\gamma_k$. For a vertex $v$ of $G$, we often denote as $p(v)$ the point of $\sigma$ that represents $v$.

## 4.1 The Drawing algorithm

At each step of Algorithm `Cater-Draw` the following three invariant properties are maintained. **Property 1**: $\gamma_k$ is a 2-page bi-chromatic topological book embedding with at most one division. **Property 2**: All the points in $NB_k$ have the same color, and each of them is accessible from one of the two pages. **Property 3**: Point $p(v_k)$ is accessible from one of the two pages.

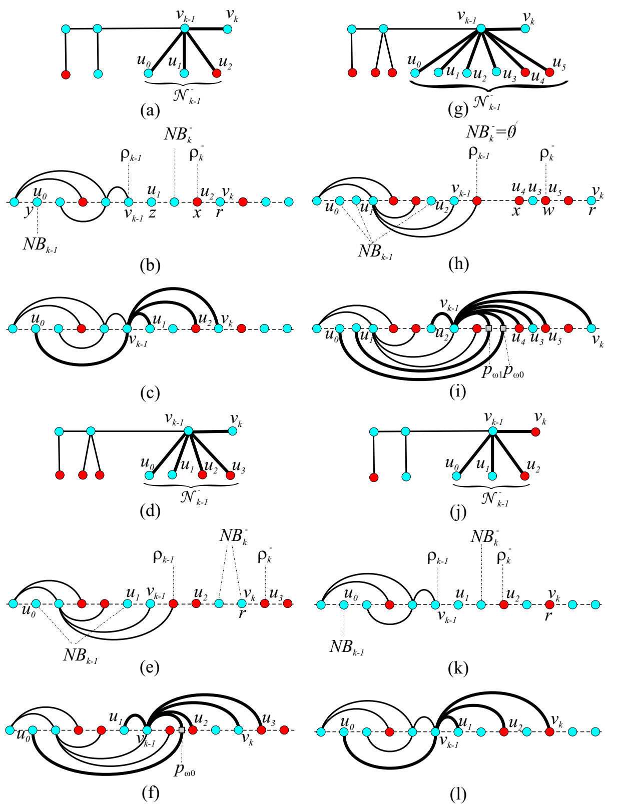


Figure 2: Different examples for Step $k$ of `Algorithm Cater-Draw`. (a),(d),(g),(j): Different examples for graph $G_k$. (b),(e),(h),(k): Mapping the vertices of $\mathcal{N}_{k-1}$ to points of $\sigma$. (c), (f), (i), (l) Drawing edges connecting $v_{k-1}$ to the vertices of $\mathcal{N}_{k-1}$.

At Step 0 of the algorithm, $\mathcal{N}_{-1}$ (i.e. vertex $v_0$) is drawn as the leftmost point $p_j$ of $\sigma$ such that $c(v_0) = c(p_j)$. The output of Step 0 is a drawing $\gamma_0$ consisting of a single vertex and for which it is immediate to see that all above invariants are satisfied. At Step $k > 0$, we assume by induction that $\gamma_{k-1}$ satisfies the invariants and show how to add $\mathcal{N}_{k-1}$ and the corresponding edges in order to compute $\gamma_k$.

The algorithm draws first the vertices of $\mathcal{N}^-_{k-1}$ and then adds $v_k$. In the description of Algorithm `Cater-Draw` we shall often refer to Figure 2 for different examples. For example, in the caterpillar of Figure 2 (a) vertex $v_{k-1}$ is blue, $\mathcal{N}^-_{k-1}$ consists of two blue vertices (vertices $u_0$ and $u_1$) and one red vertex (vertex $u_2$): Algorithm `Cater-Draw` will first draw $u_0$, $u_1$, $u_2$, and finally draw $v_k$.

Since Property 2 is satisfied by $\gamma_{k-1}$ we have that all points in $NB_{k-1}$ have the same color; we denote as $c_1$ such a color and we denote the other color as $c_2$. We draw first the vertices of $\mathcal{N}^-_{k-1}$ whose color is $c_2$, i.e. the vertices of $\mathcal{N}^-_{k-1} \cap V_{c_2}$. Assume $n_2$ is the number of such vertices (for example $n_2 = 1$ in Figure 2 (a) and $n_2 = 2$ in Figure 2 (g)). We map the vertices of $\mathcal{N}^-_{k-1} \cap V_{c_2}$ to the first $n_2$ points of color $c_2$ that are to the right of $\rho_{k-1}$; more formally, the vertices of $\mathcal{N}^-_{k-1} \cap V_{c_2}$ are mapped to the leftmost $n_2$ points of $F^{c_2}_{k-1}$. For example, the red vertex $u_2$ of $\mathcal{N}^-_{k-1}$ in Figure 2 (a) is mapped to the first red point that follows $\rho_{k-1}$ in Figure 2 (b), i.e. $u_2$ is mapped to point $x$. Vertices $u_4$ and $u_5$ of Figure 2 (g) are mapped to points $x$ and $w$ of Figure 2 (h).

We now draw the vertices of $\mathcal{N}^-_{k-1}$ that have color $c_1$. Let $n_1 = |\mathcal{N}^-_{k-1} \cap V_{c_1}|$. If $n_1 \leq |NB_{k-1}|$ the vertices of the set $\mathcal{N}^-_{k-1} \cap V_{c_1}$ are mapped to the rightmost $n_1$ points of $NB_{k-1}$ (that is, the last $n_1$ points of $NB_{k-1}$ encountered when walking along the spine of $\gamma_{k-1}$ from left to right). For example, see the drawing of vertices $u_0$, $u_1$, and $u_2$ in Figure 2 (h). If $n_1 > |NB_{k-1}|$ the vertices of $\mathcal{N}^-_{k-1}$ that have color $c_1$ are mapped to all vertices of $NB_{k-1}$ and to the $n_1 - |NB_{k-1}|$ leftmost vertices of $F^{c_1}_{k-1}$. For example in Figure 2 (a) $\mathcal{N}^-_{k-1}$ has the two blue vertices $u_0$ and $u_1$ but $NB_{k-1}$ has only one blue point (point $y$); hence we map $u_0$ to $y$ and $u_1$ to the first blue point after $\rho_{k-1}$, i.e. we map $u_1$ to point $z$.

Once all vertices of $\mathcal{N}^-_{k-1}$ have been mapped to points, Algorithm `Cater-Draw` draws $v_k$. Let $\sigma^-_k \subseteq \sigma$ be the red-blue sequence consisting of all points used to construct the drawing up to now. Let $\rho^-_k$ be the rightmost point of $\sigma^-_k$. For example, in Figure 2 (b) $\rho^-_k$ is point $x$. Let $NB^-_k$ be the set of points of $\sigma \setminus \sigma^-_k$ that are to the left of $\rho^-_k$. It can be proved that all points of $NB^-_k$ share the same color $c(NB^-_k)$. For example, in Figure 2 (b) the set $NB^-_k$ consists of one blue point. As other examples, in Figure 2 (h) $NB^-_k$ is the empty set, in Figure 2 (e), $NB^-_k$ consists of two blue points (note that all points of $NB^-_{k-1}$ belong to $\sigma^-_k$), and in Figure 2 (k) $NB^-_k$ consists of a blue point.

In order to draw $v_k$, Algorithm `Cater-Draw` distinguishes among different cases.

**Case A:** If the first point that follows $\rho^-_k$ has the same color as $v_k$, then we take it; more formally, if $c(v_k) = c(next(\rho^-_k))$ then we map $v_k$ to $next(\rho^-_k)$. For example, vertex $v_k$ of Figure 2(a) is mapped to point $r$ of Figure 2 (b).

**Case B:** If $NB^-_k \neq \emptyset$ and $c(v_k) = c(NB^-_k)$ then we map $v_k$ to the rightmost point of $NB^-_k$. For example, we map vertex $v_k$ of Figure 2 (d) to point $r$ of Figure 2 (e).

**Case C:** Otherwise, $v_k$ is mapped to the "first" point of $\sigma$ to the right of $\rho_k^-$ and having the wanted color. More formally, $v_k$ is mapped to the point $r \in \sigma$ to the right of $\rho_k^-$ such that $c(r) = c(v_k)$ and there are no other points between $\rho_k^-$ and $r$ having color $c(v_k)$. For example, in Figure 2 (h) where $NB_k^- = \emptyset$ and $c(v_k) \neq c(next(\rho_k^-))$, we have that $v_k$ is mapped to point $r$. In Figure 2 (k) $NB_k^- \neq \emptyset$ but $c(v_k) \neq c(next(\rho_k^-))$ and $c(v_k) \neq c(NB_k^-)$; hence $v_k$ is mapped to the highlighted red point $r$.

Once all vertices of $\mathcal{N}_{k-1}$ have been mapped to points, Algorithm `Cater-Draw` draws the edges connecting $v_{k-1}$ to the vertices of $\mathcal{N}_{k-1}$. We define where to draw the division vertices along the edges and to which page each edge (or portion of edge between consecutive division vertices) is assigned. We distinguish two cases.

**Case 1:** $p(v_{k-1}) = \rho_{k-1}$**.** Refer to Figures 2 (c) and (l). The edges connecting $v_{k-1}$ to the vertices mapped to points that follow $\rho_{k-1}$ (i.e. the points of of $F_{k-1}^{c_1} \cup F_{k-1}^{c_2}$) are assigned to a same page that can be arbitrarily chosen. Indeed note that since $p(v_{k-1})$ is the right-most vertex drawn so-far, $p(v_{k-1})$ is accessible from both pages and so any of the two pages can be chosen to draw edges that connect $p(v_{k-1})$ to points on its right.

Let $u$ be a vertex mapped to a point $p$ of $NB_{k-1}$; by Property 2, $p$ is accessible from a page $\pi$. Edge $(v_{k-1}, u)$ is assigned to page $\pi$. See for example edge $(v_{k-1}, u_0)$ in Figure 2 (c) and edge $(v_{k-1}, u_0)$ in Figure 2 (l).

**Case 2:** $p(v_{k-1}) \neq \rho_{k-1}$**.** Since $\gamma_{k-1}$ satisfies by inductive hypothesis the invariants, we have that by Property 3 $p(v_{k-1})$ is accessible from one page, say $\pi$. The edges connecting $v_{k-1}$ to points to the right of $\rho_{k-1}$ (i.e. to points of $F_{k-1}^{c_1} \cup F_{k-1}^{c_2}$) are assigned to $\pi$. See for example edge $(v_{k-1}, u_2)$ in Figure 2 (f) and edge $(v_{k-1}, v_k)$ in Figure 2 (i).

Let $u$ be a vertex mapped to a point $p$ of $NB_{k-1}$; by Property 2, $p$ is accessible from a page $\pi'$. There are two subcases.

**Case 2.a:** $\pi$ **coincides with** $\pi'$**.** If $\pi'$ coincides with $\pi$ then edge $(v_{k-1}, u)$ is assigned to page $\pi$. See for example edge $(v_{k-1}, u_1)$ in Figure 2 (f) and edge $(v_{k-1}, u_2)$ in Figure 2 (i).

**Case 2.b:** $\pi$ **and** $\pi'$ **are distinct.** If the two pages $\pi$ and $\pi'$ are different, it is not possible to connect $v_{k-1}$ and $u$ with a curve entirely contained in one page without creating a crossing. See for example the points $v_{k-1}$ and $u_0$ in Figure 2 (e), where $v_{k-1}$ is accessible from the top page, $u_0$ is accessible from the bottom page and any curve connecting these two points without crossing the edges of $\gamma_{k-1}$ must cross the spine.

In this case, Algorithm `Cater-Draw` splits edge $(v_{k-1}, u)$ by means of a division vertex $\omega$, then adds a dummy point $p_\omega$ between $\rho_{k-1}$ and $next(\rho_{k-1})$; finally, edge $(v_{k-1}, \omega)$ is assigned to $\pi$ and edge $(\omega, u)$ is assigned to $\pi'$. See for example the drawing of edge $(v_{k-1}, u_0)$ in Figure 2 (f).

If we have more than one vertex not accessible from $\pi$, some additional care is needed when placing the dummy points. Let $u_0, u_1, \ldots, u_{d-1}$ be the vertices of $NB_{k-1}$ that are not accessible from $\pi$ and assume that they are encountered in this order from left to right in $\sigma$. Let $\omega_i$ be the division point of edge $(v_{k-1}, u_i)$ $(i = 0, \ldots, d-1)$. In order to avoid crossings the dummy points $p_{\omega_i}$ must be placed in the order $p_{\omega_{d-1}}, \ldots, p_{\omega_1}, p_{\omega_0}$ so that edges $(u_i, \omega_i)$ do not cross each other $(i = 0, \ldots, d-1)$. See for example the edges $(v_{k-1}, u_0)$ , $(v_{k-1}, u_1)$ and the dummy points $p_{\omega_1}$, $p_{w_0}$ in Figure 2 (i): Note that if we swapped $p_{\omega_1}$ and $p_{\omega_0}$ in the drawing, the two edges $(v_{k-1}, u_0)$ and $(v_{k-1}, u_1)$ would cross each other.

It can be proved that algorithm `Cater-Draw` correctly computes a 2-page bi-chromatic topological book embedding of a 2-colored caterpillar consistent with any given red-blue sequence (see [2]). Based on this result and on Theorem 2 the following theorem holds.

**Theorem 3** *Any 2-colored caterpillar is bi-chromatic point-set embeddable with at most two bends per edge.*

It is possible to prove that if a caterpillar is properly 2-colored, the number of bends per edge can be reduced to one. This is worst-case optimal since not all properly 2-colored simple paths are bi-chromatic point-set embeddable with zero bends per edge [3].

**Theorem 4** *Any properly 2-colored caterpillar is bi-chromatic point-set embeddable with at most one bend per edge, which is worst-case optimal.*

## References

[1] E. Di Giacomo, G. Liotta, and F. Trotta. How to embed a path onto two sets of points. In *Proc. GD 2005*. Springer, 2005. To appear.

[2] E. Di Giacomo, G. Liotta, and F. Trotta. On embedding a graph on two sets of points. Technical Report RT-005-05, DIEI Università di Perugia, 2005.

[3] A. Kaneko, M. Kano, and K. Suzuki. Path coverings of two sets of points in the plane. In *Towards a Theory of Geometric Graph*, volume 342. American Math. Society, 2004.

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

[5] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graph and Combinatorics*, 17:717–728, 2001.

# Acyclic Orientation of Drawings[*]

Eyal Ackerman[†] Kevin Buchin[‡] Christian Knauer[‡] Günter Rote[‡]

**Abstract**

Given a set of curves in the plane or a topological graph, we ask for an orientation of the curves or edges which induces an acyclic orientation on the corresponding planar map. Depending on the maximum number of crossings on a curve or an edge, we provide algorithms and hardness proofs for this problem.

## 1 Introduction

Let $G$ be a *topological graph*, that is, a graph drawn in the plane such that its vertices are distinct points, and its edge set is a set of Jordan arcs, each connecting two vertices and containing no other vertex. In this work we further assume that $G$ is a *simple* topological graph, i.e., every pair of its edges intersect at most once, either at a common vertex or at a crossing point.

An *orientation* of (the edges of) a graph is an assignment of a direction to every edge in the graph. We say that an orientation is *acyclic* if the resulting directed graph does not contain a directed cycle. Finding an acyclic orientation of a given undirected (abstract) graph can be easily computed in linear time by performing a depth-first search on the graph and then orienting every *backward* edge from the ancestor to the descendent. However, is it always possible to find an orientation of the edges of a topological graph, such that a traveller on that graph will not be able to return to his starting position even if allowed to move from one edge to the other at their crossing point? Rephrasing it in a more formal way, let $M(G)$ be the planar map induced by $G$. That it, the map obtained by adding the crossing points of $G$ as vertices, and subdividing the edges of $G$ accordingly. Then we ask for an orientation of the edges of $G$ such that the induced directed planar map $M(G)$ is acyclic.

Clearly, if the topological graph is $x$-*monotone*, that is, every vertical line crosses every edge at most once, then one can orient each edge from its endpoint with the smaller $x$-coordinate towards its endpoint with the greater $x$-coordinate. Travelling on the graph under such orientation, one always increases the value of one's $x$-coordinate and therefore cannot form a directed cycle. However, not every topological graph is acyclic-orientable as Figure 1 demonstrates. Note that the degree of every vertex in this example is one. This gives rise for considering the orientation problem in the special case the degree of each vertex is one, or in other words, when one looks for an acyclic orientation of a set of *curves* embedded in the plane.

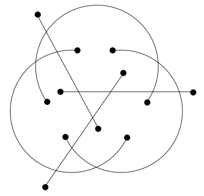


Figure 1: A non-orientable topological graph

It turns out that determining whether a topological graph (resp., a set of curves) has an acyclic orientation depends crucially on the maximum number of times an edge in the graph (resp., a curve) can be crossed. Given a (simple) topological graph $G$ on $n$ vertices, such that each edge in $G$ is crossed at most once, we show that one can find an acyclic orientation of $G$ in $O(n)$ time. When four crossings per edge are allowed, deciding whether there exists an acyclic orientation becomes NP-complete. Topological graphs with few crossings per edge were considered in several works in the literature [4, 5, 6]. For a set of $n$ curves in which each pair of curves intersects at most once and every curve is crossed at most $k$ times, we describe an $O(n)$-time orientation algorithm for the case $k \leq 3$. When $k \geq 5$ finding an acyclic orientation of the set of curves is NP-complete.

The rest of this paper is organized as follows. In Section 2 we study the problem of finding an acyclic orientation for a set of curves. Then, in Section 3 we consider the more general case where the input is a topological graph. Finally, we give some concluding remarks in Section 4, and mention a few related open problems.

## 2 Acyclic orientation of a set of curves in the plane

Throughout this paper we assume the intersections between the curves are known in advance. Given a set of curves $\mathcal{C}$, the vertices of the planar map $M(\mathcal{C})$, induced by $\mathcal{C}$, are the crossing points between the curves, while the edges of $M(\mathcal{C})$ are segments of the curves that connect two consecutive crossing points on a curve. As we have mentioned above, the maximum number of crossings per curve plays an impor-

[*]Work by the first author was done while he was visiting the Freie Universität Berlin, and was partly supported by a Marie Curie scholarship. Research by the second author was supported by the Deutsche Forschungsgemeinschaft within the European graduate program "Combinatorics, Geometry, and Computation" (No. GRK 588/2).

[†]Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel. ackerman@cs.technion.ac.il

[‡]Institute of Computer Science, Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany. {buchin|knauer|rote}@inf.fu-berlin.de

tant role when we ask for an acyclic orientation of a set of curves. If every curve is crossed at most once, then $M(\mathcal{C})$ contains no edges, and therefore any orientation of $\mathcal{C}$ is acyclic. If $\mathcal{C}$ is a set of curves with at most two crossing points per curve, then $M(\mathcal{C})$ is a union of cycles and paths and thus finding an acyclic orientation of $\mathcal{C}$ is also easy in this case. Hence, the first non-trivial case is where each curve is crossed at most three times. In this case we have:

**Theorem 1** *Let $\mathcal{C}$ be a set of $n$ curves in the plane, such that every pair of curves intersect at most once and each curve has at most three crossings. Then one can find an acyclic orientation of $\mathcal{C}$ in $O(n)$ time.*

This result is proved in Section 2.1, while in Section 2.2 we show:

**Theorem 2** *Let $\mathcal{C}$ be a set of curves in the plane, such that every pair of curves intersects at most once and each curve has at most five crossings. Then deciding whether $\mathcal{C}$ has an acyclic orientation is NP-complete.*

## 2.1 Curves with at most three crossings per curve

Let $\mathcal{C}$ be a set of $n$ curves in the plane, such that every pair of curves intersect at most once and each curve has at most three crossings. In this section we describe an algorithm for obtaining an acyclic orientation of $\mathcal{C}$. We start by constructing $M(\mathcal{C})$, the planar map induced by $\mathcal{C}$. Every connected component of $M(\mathcal{C})$ can be oriented independently, therefore we describe the algorithm assuming $M(\mathcal{C})$ is connected. Suppose $\mathcal{C}$ contains a curve $c$ which is crossed less than 3 times. By removing $c$ we obtain a set of $n-1$ curves in which there must be at least two curves (the ones crossed by $c$) which are crossed at most twice. We continue removing the curves, until none is left. Then we reinsert the curves in a reverse order (the last to be removed will be the first to be reinserted and so on). During the insertion process we reconstruct $M(\mathcal{C})$ and define a total order of its vertices. For this purpose we store the vertices of $M(\mathcal{C})$ in a data structure suggested by Dietz and Sleator [1]. This data structure supports the following operations, both in $O(1)$ worst-case time:

1. `INSERT`$(X,Y)$: Insert a new element $Y$ immediately after the element $X$.
2. `ORDER`$(X,Y)$: Compare $X$ and $Y$.

Note that by inserting $Y$ after $X$ and then switching their labels we can also use this data structure to insert a new element immediately *before* an existing element in constant time. We also keep a record of the maximal element in the order, MAX (that is, we update MAX when a new element is added after it).

We now describe the way a curve $c$ is reinserted. For every curve $c'$ that has already been added and is crossed by $c$ (recall that there are at most two such curves) we take the following actions. Let $x$ be the crossing point of $c$ and $c'$. If $c'$ has no other crossing points, then $x$ is inserted after MAX. In case $c'$ has exactly one crossing point $x'$, we insert $x$ after $x'$ when $c'$ is oriented from $x'$ to $x$, and before $x'$ otherwise. Otherwise, suppose $c'$ has two crossing points $x'_1$ and $x'_2$, such that $x'_1 < x'_2$. Then we insert $x$ before $x'_1$ if $x'_1$ is the middle point on $c'$ among the three points; after $x'_1$ if $x$ is the middle point; and after $x'_2$ if $x'_2$ is the middle point. Finally we orient $c$ arbitrarily if it has less than two crossings, or from the smaller crossing to the greater one, in case it has two crossings. We refer to the algorithm described above as Algorithm 1.

**Lemma 3** *Let $\mathcal{C}$ be a set of $n$ curves such that every curve is crossed at most three times and there is a curve that is crossed at most twice. Then the Algorithm 1 finds an acyclic orientation of $\mathcal{C}$ in $O(n)$ time.*

The more complicated case is when all the curves in $\mathcal{C}$ are crossed exactly three times. However, in this short version of the paper we only provide the general idea in this case, which is to:

1. find a set of curves $S$ that form a 'special' type of undirected cycle in $M(\mathcal{C})$;
2. orient $\mathcal{C} \setminus S$ using Algorithm 1; and
3. orient $S$ such that:
    (a) the curves in $S$ do not form a directed cycle; and
    (b) it is impossible to 'hop' on $S$ from $\mathcal{C} \setminus S$, 'travel' on $S$, and 'hop' off back to $\mathcal{C} \setminus S$.

## 2.2 Curves with at most five crossings per curve

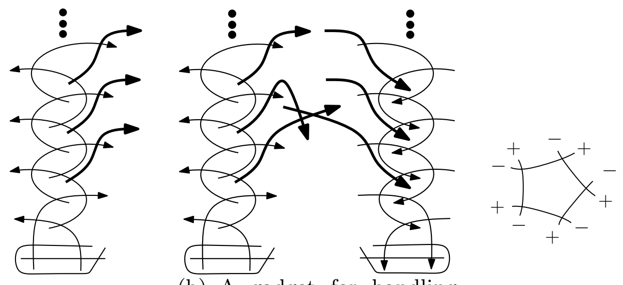
In the section we show that deciding whether there exists an acyclic orientation of a set of curves with at most 5 crossings per curve is intractable. We will reduce this problem from the NOT-ALL-EQUAL-$k$-SAT[1] ($k \geq 3$) problem which is known to be NP-complete [7].

**Proof of Theorem 2.** An acyclic orientation can be verified in polynomial time, therefore the problem is in NP. The problem is shown to be NP-hard by reduction from NOT-ALL-EQUAL-$k$-SAT to the acyclic orientation problem for $k \geq 3$.

Note that drawing the NAE problem in the plane introduces crossings between the wires. We call these crossings *extra*-crossings in order to distinguish them from the crossings between the curves. A variable is encoded as shown in Figure 2(a) where orientations correspond to Boolean signals. In any acyclic orientation all the curves drawn as arrows either have the orientation depicted or the opposite. The thick curves are used as wires and can have three further crossings. The construction uses $4+3k$ curves to generate $k$ wires.

Figure 2(b) shows the encoding of a NOT gate. It uses two wires from one variable and one from the other. The latter is used to propagate the signal across an extra-crossing without introducing a sixth crossing on a curve (note that this wire has only four

[1]NOT-ALL-EQUAL-$k$-SAT ($k \geq 3$) is given by a collection of clauses, each containing exactly $k$ literals. The problem is to determine whether there exists a truth assignment such that each clause has at least one true and one false literal.



(a) A variable (b) A gadget for handling negation and extra-crossings (c) A clause

Figure 2: A reduction from NAE-$k$-SAT to orientation of curves with at most 5 crossings per curve.

crossings). Using this gadget a signal is negated before and after the extra-crossing, thus preventing the introduction of new cycles through the extra-crossing point.

The encoding of a clause with $k$ literals is done by $k$ curves forming a $k$-gon, as shown in Figure 2(c) for $k = 5$. A wire enters at the plus or at the minus sign depending on whether its corresponding literal in the clause in negated. The edges of the $k$-gon form a directed cycle if and only if all the literals of the clause are true or all are false. A solution to the Not-All-Equal-$k$-SAT problem will therefore yield an acyclic orientation of the curves. Conversely, if there is no solution, any orientation will either have a cycle at a clause encoding, or have outgoing edges at a variable encoding with different orientations, forcing a cycle within the variable. □

## 3 Acyclic orientation of topological graphs

Given a topological graph in which no edge is crossed, one can use the simple algorithm for abstract graphs described in the Introduction to find an acyclic orientation. Thus, the first non-trivial case is when every edge is crossed as most once.

### 3.1 Topological graphs with at most one crossing per edge

**Theorem 4** *Let $G$ be a simple topological graph on $n$ vertices in which every edge is crossed at most once. Then $G$ has an acyclic orientation. Moreover, such an orientation can be found in $O(n)$ time.*

Before proving Theorem 4 we recall a basic term from graph theory [3].

**Definition 1** *Given a biconnected graph $G = (V, E)$ and an edge $(s, t) \in E$, an* st-numbering *(or* st-ordering*) of $G$ is a bijection $\ell : V \to \{1, 2, \dots, |V|\}$ such that: (a) $\ell(s) = 1$; (b) $\ell(t) = |V|$; and (c) for every vertex $v \in V \setminus \{s, t\}$ there are two edges $(v, u), (v, w) \in E$ such that $\ell(v) < \ell(u)$ and $\ell(v) > \ell(w)$.*

Given an st-numbering we will not make a distinction between a vertex and its st-number. An st-ordering of a graph $G$ naturally defines an orientation of the edges of $G$: direct every edge $(u, v)$ from $u$ to $v$ if $u < v$ and from $v$ to $u$ otherwise. The proof of the next lemma is omitted due to lack of space.

**Lemma 5** *Let $G = (V, E)$ be a plane biconnected multi-graph such that $|V| > 2$, and denote by $G'$ the directed planar graph defined by some st-numbering of $G$. Let $f$ be a face of $G$ (and $G'$), and denote by $G'_f$ (resp., $G_f$) the graph induced by the edges of $G'$ (resp., $G$) bounding $f$. Then $G'_f$ has exactly one source and one sink.*

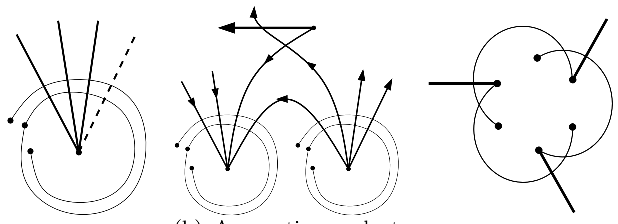
**Algorithm 3** Acyclic orientation of a topological graph with at most one crossing per edge

**Input**: A topological graph $G$ with at most one crossing per edge.
**Output**: An acyclic orientation of $G$.

```
 1: for each pair of crossing edges $(a, b)$ and $(c, d)$ do
 2:    add the edges $(a, c), (a, d), (b, c)$, and $(b, d)$;
 3: end for
 4: compute the biconnected components of the new
    graph;
 5: for each biconnected component $C$ do
 6:    delete all pairs of crossing edges in $C$;
 7:    find an st-numbering of the remaining graph;
 8:    reinsert all pairs of crossing edges in $C$;
 9:    orient each edge of $C$ according to the st-
       numbering;
10: end for
11: remove the edges added in line 2;
```

**Proof of Theorem 4.** Let $G$ be a simple topological graph in which every edge is crossed at most once. Denote by $n$ the number of vertices in $G$, and by $m$ the number of its edges. We will show that Algorithm 3 computes an acyclic orientation of $G$. Denote by $G'$ the graph obtained after adding the edges in lines 1–3. Note that it is always possible to add the edges listed in line 2 without introducing new crossings. After this step the vertices of each crossing pair of edges lie on a simple 4-cycle. It is enough to verify that each biconnected component of $G'$ is acyclicly oriented, since (a) every simple cycle in the underlying abstract graph is contained entirely in some biconnected component; and (b) the crossings do not introduce any interaction between different biconnected components, as all the vertices of a crossing pair of edges lie on a simple 4-cycle and therefore are in the same biconnected component. Thus, for the rest of the proof we assume $G'$ is biconnected. We denote by $G''$ the graph obtained by removing all the pairs of crossing edges. Removing a chord from a cycle does not affect the connected components of a graph, thus $G''$ is biconnected. Therefore, in line (7) an st-numbering of $G''$ is indeed computed.

Clearly, one can obtain an acyclic orientation of an abstract graph by numbering the vertices of the graph and directing every edge from its endpoint with the smaller number to its endpoint with the larger number. Therefore, it is enough to verify that the crossing



(a) A variable (b) A negation gadget with an edge (wire) crossed only once so far (c) A clause gate

Figure 3: A reduction from NAE-$k$-SAT to acyclic orientation of a topological graph with at most 4 crossings per edge.

points do not introduce a *bad* "shortcut", that is a path from a vertex $u$ to a vertex $v$ such that $v < u$. Let $((a, b), (c, d))$ be a pair of crossing edges. Denote by $f$ the 4-face $a - c - b - d - a$ of $G''$. According to Lemma 5 the digraph induced by $f$ and the computed st-numbering has only one source and sink. Therefore, we have to consider only two cases based on whether the sink and the source are adjacent in $f$. One can easily verify by inspection that in both cases no bad shortcut is formed. Thus Algorithm 3 produces an acyclic orientation.

Note that Algorithm 3 can be implemented to run in time linear in the number of vertices: Finding the biconnected components of a graph takes $O(n + m)$ time [2], as does the computation of an st-numbering [3]. Therefore the overall running time is $O(n + m)$, however the maximum number of edges in a topological graph in which every edge is crossed at most once is $4n - 8$ [6]. □

## 3.2 Topological graphs with at most four crossings per edge

**Theorem 6** *Let $G$ be a simple topological graph on $n$ vertices in which every edge is crossed at most once, and each curve has at most four crossings. Then deciding whether $G$ has an acyclic orientation is NP-complete.*

**Proof** (sketch). As for the case of a set of curves, the reduction is done from Not-All-Equal-$k$-SAT. The gadgets we use appear in Figure 3. □

## 4 Discussion

We considered the problem of finding an acyclic orientation for a given topological graph or a set of curves in the plane. For topological graphs with at most one crossing per edge we showed an algorithm for finding an acyclic orientation in linear time. It follows from our results that when the maximum crossing per edge is at least four, deciding whether an acyclic orientation of the graph exists is NP-complete. An obvious open question is what happens when the maximum number of crossings per edge is two or three. A non-orientable topological graph with at most three crossings per edge can be constructed by combining the gadgets shown in Figure 3(a) (without the dashed wire) and in Figure 3(c). However, deciding whether a topological graph with at most three crossings per edge has an acyclic orientation is open. The situation is worse for topological graphs with at most two crossings per edge: So far we were unable to find an example which has no acyclic orientation, or to prove that every such graph is acyclic-orientable.

A special case is where all the vertices in the topological graph have degree 1. This case corresponds to asking the acyclic orientation question for a set of curves. Clearly, if the problem can be solved (or decided) for topological graphs with at most $k$ crossings per edge, then it can be solved for curves with at most $k$ crossings per curve. It would be interesting to determine whether there is a construction that provides a reduction from topological graphs with at most $k$ crossings per edge to a set of curves with at most $k'$ crossings per curve.

For curves with at most three crossings per curve we provided a linear time algorithm that finds an acyclic orientation. For five crossings per curve we showed that the problem becomes NP-complete. A set of curves in which every curve is crossed at most four times might not have an acyclic orientation, as Figure 1 implies. However, the decision problem for such sets of curves is also open. Two other interesting open questions are: (1) What happens if we only require acyclic *faces*? and (2) What happens if we look for an orientation such that for every pair of vertices, $u, v$, in the induced planar map there is a directed path from $u$ to $v$ or vice versa?

**Acknowledgements** We thank Michel Pocchiola for suggesting a problem that led us to study the questions discussed in this paper. We also thank Scot Drysdale, Frank Hoffmann, and Klaus Kriegel for helpful discussions.

## References

[1] P. Dietz and D. Sleator, Two algorithms for maintaining order in a list, *Proc. 19th Ann. ACM Symp. on Theory of Computing* (STOC), NYC, NY, 1987, 365–372.

[2] S. Even, *Graph Algorithms*, Computer Science Press, 1979.

[3] S. Even and R. E. Tarjan, Computing an st-numbering, *Theoretical Computer Science*, 2(3):339–344, 1976.

[4] A. Grigoriev and H. L. Bodlaender, Algorithms for graphs embeddable with few crossings per edge, *Proc. 15th Int. Symp. on Fundamentals of Computation Theory* (FCT), Lübeck, Germany, *Lecture Notes in Computer Science*, volume 3623, Springer, 378–387, Sep. 2005.

[5] J. Pach, R. Radoicic, G. Tardos, and G. Tóth, Improving the crossing lemma by finding more crossings in sparse graphs, *Proc. 20th ACM Symp. on Computational Geometry* (SoCG), Brooklyn, NY, 2004, 68–75.

[6] J. Pach and G. Tóth, Graphs drawn with few crossings per edge, *Combinatorica*, 17(3):427–439, 1997.

[7] T. J. Schaefer, The complexity of satisfiability problems, *Proc. 10th Ann. ACM Symp. on Theory of Computing* (STOC), San Diego, CA, 1978, 216–226.

# An homotopy theorem for arrangements of double pseudolines

Luc Habert Michel Pocchiola*

## Abstract

We define a *double pseudoline* as a simple closed curve in the open Möbius band homotopic to the double of its core circle, and we define an *arrangement of double pseudolines* as a collection of double pseudolines such that every pair crosses in 4 points – the crossings being transversal – and induces a cell decomposition of the Möbius band whose 2-dimensional cells are 2-balls, except the unbounded cell which is a 2-ball minus a point. Dual arrangements of boundaries of collection of pairwise disjoint 2-dimensional closed bounded planar convex sets are examples of arrangements of double pseudolines. We show that every pair of simple arrangements of double pseudolines is connected by a sequence of triangle-switches and that every simple arrangement of double pseudolines has a *representation* by a configuration of pairwise disjoint disks in the plane with pseudoline double tangents. This shows in particular that any double-permutation sequence of J.E. Goodman and R. Pollack (SoCG'05 page 159, [2]) has a representation by a configuration of pairwise disjoint disks in the plane with pseudoline double tangents. We also present some enumeration results for our arrangements, and a property of their subarrangements.

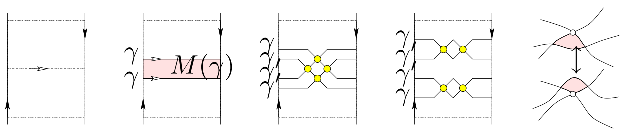


Figure 1: The Möbius band and its core circle, a (monotone) double pseudoline $\gamma$ and the Möbius band $M(\gamma)$ bounded by $\gamma$, an arrangement of two double pseudolines $\gamma$ and $\gamma'$, a collection of two double pseudolines with 4 crossing points but which is not an arrangement because the cell intersection of the associated Möbius bands is not a 2-ball, and a triangle-switch.

**1. Arrangements of double pseudolines in the Möbius band.** Let $\mathbb{M}$ be the open Möbius band, say quotient of $\mathbb{R}^2$ under the map $\iota : \mathbb{R}^2 \to \mathbb{R}^2$ that assigns to the pair $(\theta, u)$ the pair $(\theta + \pi, -u)$, and let $c : [0,1] \to \mathbb{M}$, $c(t) = (\pi t, 0)$, be its core circle. A *pseudoline* in $\mathbb{M}$ is a simple closed path in $\mathbb{M}$ homotopic to its core circle and a *double pseudoline* in $\mathbb{M}$ is a simple closed path in $\mathbb{M}$ homotopic to the double $cc$ of its core circle. Boundary curves of tubular neighborhoods of pseudolines are examples of double pseudolines. We define an *arrangement of double pseudolines* in $\mathbb{M}$ as a finite collection $\Gamma$ of double pseudolines in $\mathbb{M}$ such that every pair of elements of $\Gamma$ crosses in 4 points – the crossings being transversal – and induces a cell decomposition of $\mathbb{M}$ whose 2-dimensional cells are 2-balls, except the unbounded cell which is a 2-ball minus a point, and we define the *chirotope* $\chi_\Gamma$ of $\Gamma$ as the map that assigns to each $\gamma \in \Gamma$ and to each ordered triple $u, v, w$ of vertices lying on $\gamma$ of the cell decomposition $X_\Gamma$ of $\mathbb{M}$ induced by the elements of $\Gamma$ the value $+1$ if walking along the curve $\gamma$ we encounter the vertices in cyclic order $uvwuvw\cdots$; $-1$ otherwise. An arrangement of double pseudolines is called *simple* if exactly two double pseudolines meet at every vertex of the induced cell decomposition of $\mathbb{M}$. A simple arrangement of double pseudolines is called *thin* if there is no vertex of the induced cell decomposition lying in the interiors of the closed Möbius bands bounded by the double pseudolines. Thin arrangements are obtained from simple (finite) arrangements of pseudolines by replacing the pseudolines by the boundary curves of suitable tubular neighborhoods. Collections of dual curves[1] of boundaries of pairwise disjoint 2-dimensional closed bounded planar convex sets are examples of arrangements of double pseudolines; these arrangements are simple and thin under the additional assumption that there is no line transversal to three convex sets; these arrangements are also *monotone* with respect to the core circle in the sense every meridian $\theta = c$ of the Möbius band crosses every double pseudoline exactly twice. Finally we observe that, as in the case of arrangements of pseudolines, the set of arrangements of double pseudolines is stable by triangle-switch. The main result of the paper is the following.

**Theorem 1** *Let $\Gamma$ be a simple arrangement in $\mathbb{M}$ of double pseudolines, $X$ the induced cell decomposition of $\mathbb{M}$, and $\gamma \in \Gamma$. Assume that there is a vertex of $X$ lying in the interior of the Möbius band $M(\gamma)$ bounded by $\gamma$. Then there is a triangular face of $X$*

*Department of Computer Science, Ecole normale supérieure, Paris, {Luc.Habert,Michel.Pocchiola}@ens.fr

[1]The dual of a planar smooth curve is the curve in the space of undirected lines of the plane of the tangent lines to the curve. We identify the space of undirected lines of the plane with the Möbius band $\mathbb{M}$ via the map that assigns to the pair $(\theta, u)$ the line with equation $u - x \sin\theta + y \cos\theta = 0$.

*included in $M(\gamma)$ with a side supported by $\gamma$.* □

Therefore triangle-switches are always possible for non thin arrangements of double pseudolines and, consequently, every arrangement of double pseudolines is homotopic to a "core" thin one; furthermore since thin arrangements are arrangements of boundary curves of tubular neighborhoods of simple arrangements of pseudolines and since Ringel's homotopy theorem for arrangements of pseudolines [1, page 267] [5, 6] asserts that the set of simple arrangements of pseudolines is connected by triangle-switches we get a similar result for the set of arrangement of double pseudolines on a fixed number of double pseudolines. We summarize.

**Corollary 2** *Let $\Gamma$ be a simple arrangement of double pseudolines. Then there exists an homotopy $\Gamma_t$, $t \in [0,1]$, of $\Gamma = \Gamma_0$ onto a thin arrangement of double pseudolines $\Gamma_1$ such that $M(\gamma_t) \supseteq M(\gamma_{t'})$ for all $t \leq t'$. (Simplicity is of course not maintained during the homotopy.)* □

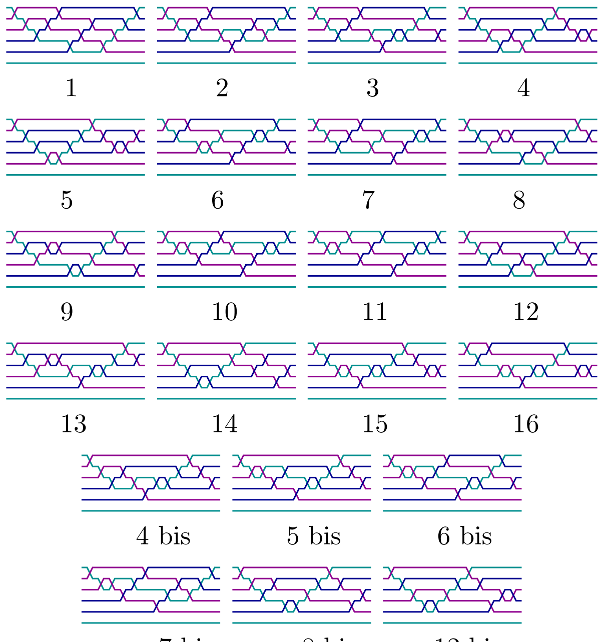
**Corollary 3** *Let $\Gamma$ and $\Gamma'$ be two arrangements of $n$ double pseudolines. Then $\Gamma$ and $\Gamma'$ are homotopic.* □

**2. Combinatorial equivalence and enumeration results.** For $\Gamma$ an arrangement of two double pseudolines $\gamma$ and $\gamma'$ we set $v(\gamma, \gamma')$ to be the linear sequence of vertices of $X_\Gamma$ encountered when walking along the curve $\gamma$ starting from the source of the 2-cell intersection of the Möbius bands $M(\gamma)$ and $M(\gamma')$ associated with $\gamma$ and $\gamma'$. Let $\Gamma$ and $\Gamma'$ be two arrangements of $n$ double pseudolines and let $\varphi : \Gamma \to \Gamma'$ be a bijection. We denote by $\tilde{\varphi} : \Gamma \cup X_0 \to \Gamma' \cup X'_0$ the canonical extension of $\varphi$ to the sets of vertices of $X_\Gamma$ and $X_{\Gamma'}$ defined by the condition that $\tilde{\varphi}$ assigns to the sequence $v(\gamma_1, \gamma_2)$ the sequence $v(\varphi\gamma_1, \varphi\gamma_2)$ for all $\gamma_1, \gamma_2 \in \Gamma$. We say that $\Gamma$ and $\Gamma'$ are *$\varphi$-equivalent* if $\chi_{\Gamma'} \circ \tilde{\varphi} = \chi_\Gamma$, that $\Gamma$ and $\Gamma'$ are *combinatorially equivalent* if $\Gamma$ and $\Gamma'$ are $\varphi$-equivalent for some bijection $\varphi : \Gamma \to \Gamma'$, and that $\Gamma$ and $\Gamma'$ are *combinatorially equivalent up to (global) reorientation* if $\Gamma$ is equivalent to $\Gamma'$ or to $\Gamma'^{-1}$. Finally we say that two ordered arrangements of $n$ double pseudolines are *combinatorially equivalent* if they are $\varphi$-equivalent for the bijection $\varphi$ induced by the orderings. We denote by $a_n$ the number of classes of ordered simple arrangements of $n$ double pseudolines, and by $b_n$ et $c_n$ the numbers of classes of simple arrangements of $n$ double pseudolines under the combinatorial equivalence and combinatorial equivalence up to (global) reorientation.

Our main result provides an algorithm to enumerate the combinatorial equivalence classes of ordered arrangements and arrangements of $n$ double pseudolines by a traversal of the triangle-switch graph of double pseudoline arrangements. We have implemented this algorithm. Preliminary results are reported in the following table that confirms the result reported in [8] concerning the value of $c_4$.

| $n$ | 2 | 3 | 4 |
|---|---|---|---|
| $a_n$ | 1 | 118 | – |
| $b_n$ | 1 | 22 | 22620 |
| $c_n$ | 1 | 16 | 11502 |

The numbers $a_n, b_n$ et $c_n$ are in $2^{\Theta(n^2)}$ since the cell decomposition $X_\Gamma$ induced by an arrangement $\Gamma$ of size $n$ is constructable in time $O(n^2)$ using an incremental ramdomized algorithm [7]. Lower bounds were established (even for dual arrangements of convex sets) in [3, 4]. Representatives of the $22 = 16 + 6$ classes of combinatorially equivalent arrangements of three double pseudolines are depicted in the figure below. The 10 arrangements numbered 1,2,3,9,10,11,12,13,14,15 and 16 are combinatorially equivalent to their (global) reoriented versions. The reoriented versions of the arrangements numbered 4,5,6,7,8 and 12 are numbered 4 bis, 5 bis, 6 bis, 7 bis, 8 bis and 12 bis.



**3. Representation by arrangements of disks.** We use the following terminology. A *disk* is 2-dimensional bounded closed simply connected subset of the affine oriented plane. A *tangent* to a disk is a pseudoline that intersects the disk at a sole boundary point; in particular a disk is included in the closure of one of the two connected components of the complement of any of its tangents (half-plane for short). An *arrangement* of disks is a finite collection $\Delta$ of pairwise disjoint disks equipped with a map that assigns to each unordered pair $o, o'$ of disks a set $L(o, o')$ of four double tangents to $o$ and $o'$ such that the whole set of pseudolines $L(\Delta)$ union of the $L(o, o')$'s is an affine arrangement of pseudolines and such that the intersection of a disk and a line is connected. We denote by $L(o)$ the set of pseudolines of $L(\Delta)$ that are tangents to $o$. An arrangement of disks is called *simple* if every tangent of the arrangement is tangent to exactly two disks. Wlog we will assume that a touching point between a double tangent and a disk lies on exactly one double tangent. The *chirotope* of an arrangement of disks $\Delta$ is the map $\chi_\Delta$ that assigns to each disk $o$ and to each ordered triple $u, v, w$ of tangents to $o$ the value $+1$ if walking counterclockwise around the boundary of the disk $o$ we encounter the tangents in cyclic order $uvwuvw\cdots$; -1 otherwise.

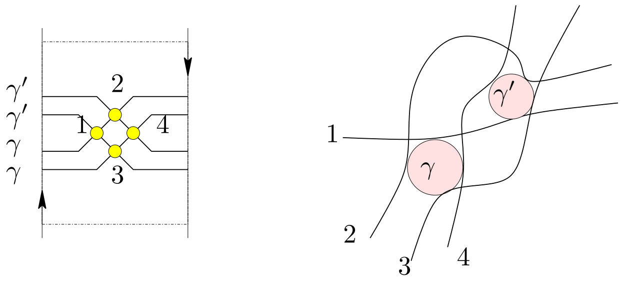


Figure 2: Local representation : the double pseudolines $\gamma$ and $\gamma'$ are represented by the disks labelled $\gamma$, $\gamma'$ and the 4 vertices numbered 1,2,3, and 4 of the induced cell decomposition of $\mathbb{M}$ are represented by the pseudoline double tangents numbered 1,2,3 and 4.

Let now $\Gamma$ be an arrangement of double pseudolines. An arrangement $\Delta$ of disks is called a *representation* of $\Gamma$ if $\Delta$ and $\Gamma$ have the same chirotope, i.e., there is a bijection $\varphi : \Gamma \to \Delta$ such that its extension $\varphi : X^0 \to L$ between the set $X^0$ of vertices of $X_\Gamma$ and the set of double tangents $L$ of $\Delta$ defined in Figure 2 carries the chirotope of $\Gamma$ onto the chirotope of $\Delta$, i.e., $\chi_\Delta \circ \varphi = \chi_\Gamma$.

**Theorem 4** *Every simple arrangement of double pseudolines is representable by an arrangement of disks.*

**Proof.** (Sketch.) Since representable arrangements of double pseudolines exist it is sufficient, thanks to Theorem 1, to show that the property to have a representation is maintained during a triangle-switch operation. So let $\Gamma$ be a simple arrangement of double pseudolines represented by an arrangement $\Delta$ of disks, i.e., there exists a bijection $\varphi : \Gamma \to \Delta$ such that $\chi_\Delta \circ \varphi = \chi_\Gamma$. Let $\sigma$ a triangular face $\alpha\beta\gamma$ of $X$ whose edges $\alpha\beta$, $\beta\gamma$ and $\alpha\gamma$ are supported by the double pseudolines $U, V$ and $W$. Using a simple perturbation argument it is sufficient to prove that the (non simple) arrangement of double pseudolines $\Gamma'$ obtained by collapsing the triangular face $\alpha\beta\gamma$ to a single point is representable by an arrangement of disks $\Omega'$. The disks and double tangents corresponding to $U, V, W, \alpha, \beta, \gamma$ will be simply denoted $U^*, V^*, W^*, \alpha^*, \beta^*$ and $\gamma^*$. We orient the double tangent $\alpha^*$ from $V^*$ towards $U^*$ and $\beta^*$ and $\gamma^*$ from $U^*$ towards $W^*$. In that case a simple analysis shows that the relative positions of the disks and the double tangents are as indicated in the top left part of Figure 3. (There are $2^3 = 8$ combinatorially different cases which correspond to the choice of the positions of the disks with respect to the double tangents.)

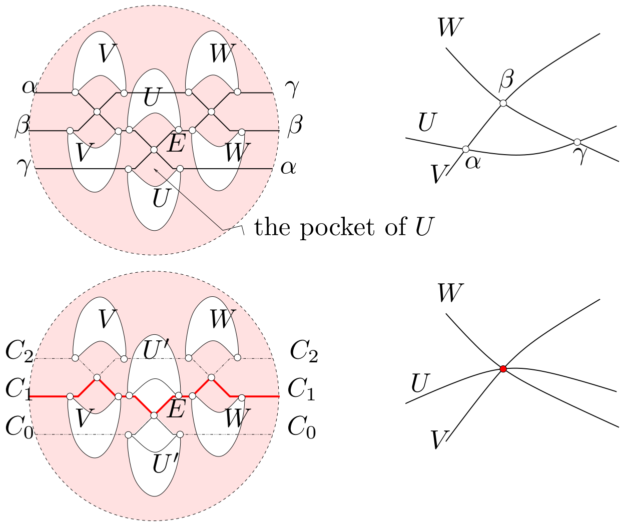


Figure 3: The three double pseudolines $\alpha^*, \beta^*$ and $\gamma^*$ are replaced by the 1-level of their arrangement.

Let $C_0, C_1$ and $C_2$ be the 0-, 1- and 2-level of the arrangement of the pseudolines $\alpha^*, \beta^*$ and $\gamma^*$. We set $L' = L \setminus \{\alpha^*, \beta^*, \gamma^*\} \cup \{C_1\}$ and we define $U'$ to be the union of the disk $U^*$ and the "pocket" delimited by the contact points with $U^*$ of the double tangents $\alpha^*$ and $\gamma^*$ and their intersection point $E$ and we define $U''$ to be a slight perturbation of $U'$ so that its intersection with $C_1$ reduces to the vertex $E$ (this pertubation has do be done only if $U^*$ lies on the right sides of $\gamma^*$ and $\alpha^*$); similar constructions are done with the disks $V^*$ and $W^*$. We set $\Omega' = \Omega \setminus \{U, V, W\} \cup \{U'', V'', W''\}$. One can check that $L'$ is an arrangement of pseudolines and that the collection of disks $\Omega'$ equipped with the set $L'$ is an arrangement of disks whose chirotope is combinatorially equivalent to the chirotope of $\Gamma'$. $\square$

## 4. Wiring diagrams and double-permutation sequences.

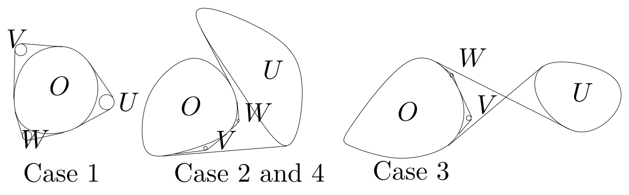
Since the triangle-switch operation can be implemented to preserve the monotonicity of the curves with respect to the core circle we see that every simple arrangement $\Gamma$ of double pseudolines is combinatorially equivalent to a monotone one. Therefore one can not only speak of the source, sour$(\sigma)$, and the sink, sink$(\sigma)$, of every 1-cell or 2-cell $\sigma \in X_\Gamma$ but the transitive closure of the covering relations sour$(\tilde{\sigma}) \prec \tilde{\sigma} \prec$ sink$(\tilde{\sigma})$ defined on the lifts of the cells of $X_\Gamma$ in the universal covering $p : \mathbb{R}^2 \to \mathbb{M}$ of $\mathbb{M}$ is a well-defined *partial order* on $\tilde{X}_\Gamma$. The following theorem provides a complete description of the set of monotone arrangements combinatorially equivalent to a given simple arrangement and gives an interpretation in terms of representation by arrangements of disks (answering positively a question set in [2, Remark 20] regarding the realizability of a double-permutation sequence by an arrangement of disks).

**Theorem 5** *Let $\Gamma$ be a monotone arrangement of $n$ double pseudolines in $\mathbb{M}$ and assume that walking along the core circle we encounter the vertical projections $v_i'$ of the vertices $v_i$ of $X_\Gamma$ in the circular order $v_1'v_2' \dots v_{2n(n-1)}'$. Then (1) $v_1v_2 \dots v_{2n(n-1)}$ is the projection on $\mathbb{M}$ of a linear extension of the poset $(\tilde{X}_\Gamma^0, \prec)$ compatible with $\iota$, i.e., if $v$ precedes $w$ then $\iota(v)$ precedes $\iota(w)$; (2) $\Gamma$ is representable by an arrangement $\Delta$ of $n$ disks such that the circular ordering of the corresponding double tangents with respect to the line at infinity corresponds to the circular ordering of the vertices.* □

**5. On the chirotopes of subarrangements.** By definition the chirotope of an arrangement of $n$ disks depends only on its $\binom{n}{4}$ subarrangements of 4 disks. Since chirotopes of collections of points depends only by definition of the collection of chirotopes of subsets of three points it is natural toask if the same holds for arrangements of disks. The answer is yes. This result was conjectured[2] to be true for stretchable arrangements in [3, Theorem 3.] and checked using the exhaustive list of 11502 arrangements on 4 disks generated by triangle-switches starting from a stretchable arrangement by F. Torossian [8], ten years ago. We provide a direct proof below; this direct proof confirms partially the validity of our implementation.

**Theorem 6** *The chirotope of an arrangement of $n$ disks depends only on the $\binom{n}{3}$ chirotopes of its subarrangements of 3 disks.*

**Proof.** (Sketch.) Let $O, U, V, W$ be 4 disks. We denote by $\chi$ the chirotope and $\chi^3$ its restriction to triple of disks. We write $\mathrm{L}(X)$ for the set of double tangents to $O$ and $X$, and we write $(a, b)$ for the set of $x$ such that $\chi_O(a, x, b)$. Let $u, u' \in \mathrm{L}(U)$ and let $x \in \mathrm{L}(U, V, W)$. Clearly one can decide if $\chi(u, x, u')$ using only $\chi^3$. A pair $(u, u')$ of bitangents is said to separate the pair $(x, x')$ if $u, u', x, x'$ appear in the cyclic order $uxu'x'$ around the disk $O$. Assume that a pair $(u, u')$ of elements of $\mathrm{L}(U)$ separates a pair $(x, x')$ of elements of $\mathrm{L}(V, W)$. In that case a triplet $a, b, c$ of elements of $\mathrm{L}(V, W)$ lying in the interval $(u, u')$ appears in the linear order $abc$ in the interval $(u, u')$ iff. $x', a, b, c$ appear in the cyclic order $x'abc$ around the disk $o$. So it remains to examine the case where no pair of elements of $\mathrm{L}(U)$, $\mathrm{L}(V)$ and $\mathrm{L}(W)$ separates a pair of elements of $\mathrm{L}(V, W)$, $\mathrm{L}(W, U)$ and $\mathrm{L}(U, V)$. In that case $\chi(u, v, w)$ is independant of the choice of $u \in \mathrm{L}(U), v \in \mathrm{L}(V)$ and $w \in \mathrm{L}(W)$. We will simply write $\chi(U, V, W)$.



Let 1,2,3 and 4 be the consecutive elements of $\mathrm{L}(U)$ on $O$ where 1 is the right-left double tangent oriented from $O$ toward $U$. We write $\mathrm{I}_1(U), \mathrm{I}_2(U), \mathrm{I}_3(U), \mathrm{I}_4(U)$ for the interval $(3, 4), (4, 1), (1, 2), (2, 3)$. It remains four cases to examine (cf. figure above):
**case 1.** $\mathrm{L}(V, W) = \mathrm{I}_4(U)$. In that case $\chi(U, V, W)$ iff $U, V$ et $W$ appear in this order when walking counterclockwise around the boundary of their convex hull.
**case 2.** $\mathrm{L}(V, W) = \mathrm{I}_1(U)$. In that case $\chi(U, V, W)$ iff $W$ lies in the interior of the convex hull of $U$ and $V$;
**case 3.** $\mathrm{L}(V, W) = \mathrm{I}_2(U)$. In that case $\chi(U, V, W)$ iff $U, W, V$ appear in this in order when walking counterclockwise on the boundary of their convex hull.
**case 4.** $\mathrm{L}(V, W) = \mathrm{I}_3(U)$. similar to case 2 modulo a reorientation of the plane.

Furthermore in each of these four cases the chirotope is unique up to the permutation of $V$ and $W$. □

## References

[1] A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. M. Ziegler. *Oriented Matroids.* Cambridge University Press, Cambridge, 1993.

[2] J.E. Goodman and R. Pollack. The combinatorial encoding of disjoint convex sets in the plane, and a generalization of the edelsbrunner-sharir transversal theorem, 2004.

[3] M. Pocchiola and G. Vegter. Order types and visibility types of configurations of disjoint convex plane sets (extended abstract). Technical Report 94-4, Labo. Inf. Ens, 45 rue d'Ulm 75230 Paris, France, January 1994.

[4] M. Pocchiola and G. Vegter. Pseudo-triangulations: Theory and applications. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 291–300, 1996.

[5] G. Ringel. Teilungen der ebene durch geraden oder topologishe geraden. *Math. Zeitschrift*, 64:79–102, 1956.

[6] G. Ringel. Über geraden in allgemeiner lage. *Elemente der Math.*, 12:75–82, 1957.

[7] R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 37–68. Springer-Verlag, 1993.

[8] F. Torossian. Enumération des arrangements duaux de quatre convexes du plan. DEA, Ecole Normale Supérieure, July 1996.

[2] Claimed without proof to be more honest!

# Tight planar packings of two trees

Yoshiaki Oda* Katsuhiro Ota†

**Abstract**

When we consider an embedding of graphs into a plane, it would be nice if it does not intersect internally since the embedding simply shows us the structure of graphs. It is easy to embed a tree into a plane with non-self-intersections. If we embed two or more trees into a plane with non-self-intersections, what occurs? In this paper, we consider embeddings of two trees into a plane with using same vertices, sharing no edges and no-intersections. We prove that a non-star tree and a non-star caterpillar can be embedded into a plane satisfying the above conditions. It is one of the special cases of a conjecture by Garcia et al. [1].

## 1 Introduction

In this paper, we deal with finite undirected graphs with neither loops nor multiple edges. The following problem is well-known.

For given graphs $G, T_1, \cdots, T_k$, does $G$ contain subgraphs isomorphic to $T_1, \cdots, T_k$ and pairwise edge disjoint?

This problem is NP-complete since if we suppose that $G$ is a complete graph $K_n$ and $k = 2$ then the problem is equivalent to SUBGRAPH ISOMORPHISM which is known to be NP-complete [2].

We say that graphs $T_1, \cdots, T_k$ can be *packed into a graph* $G$ or *there exists a packing of* $T_1, \cdots, T_k$ *into* $G$ if $G$ contains subgraphs isomorphic to $T_1, \cdots, T_k$ and pairwise edge disjoint. Moreover, if it satisfies $|V(T_1)| = \cdots = |V(T_k)| = |V(G)|$, we say that $T_1, \cdots, T_k$ can be *tightly packed into* $G$ or *there exists a tight packing of* $T_1, \cdots, T_k$ *into* $G$. A star with $n$ vertices means a complete bipartite graph $K_{1,n-1}$.

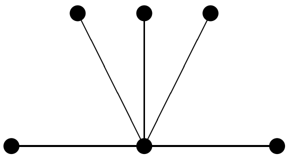


Figure 1: A star with 6 vertices ($K_{1,5}$)

*Department of Mathematics, Keio University, oda@math.keio.ac.jp

†Department of Mathematics, Keio University, ohta@math.keio.ac.jp

Hedetniemi et al. [3] proved the following theorem.

**Theorem 1 (Hedetniemi et al. [3])** *Let $T_1$ and $T_2$ be non-star trees with $n$ vertices. Then, there exists a tight packing of $T_1$ and $T_2$ into a complete graph $K_n$.*

Here we suppose that $G$ is a plane graph. Garcia et al. [1] gave the following conjecture.

**Conjecture 1 (Garcia et al. [1])** *Let $T_1$ and $T_2$ be non-star trees with $n$ vertices. Then, there exists a simple plane graph $G$ such that $T_1$ and $T_2$ can be tightly packed into $G$.*

If the above plane graph exists, we say that *there exists a tight planar packing of $T_1$ and $T_2$*. We observe that such a packing is equivalent to a planar drawing of $T_1$ and $T_2$ without sharing edges. In this case, we obtain a simultaneous drawing of $T_1$ and $T_2$ on the same vertex set.

The following observations are fundamental.

**Observation 1** *Let $T_1$ be a star and $T_2$ be a tree with $n$ vertices. Then, there does not exist any tight planar packing of $T_1$ and $T_2$.*

**Proof.** Suppose that $T_1$ and $T_2$ can be packed into a simple plane graph $G$. We focus on the center vertex $v$ of the star $T_1$. Any edge which is incident with the vertex $\bar{v}$ of $G$ corresponding to $v$ must be used for $T_1$. Hence, we can assign no vertex of $T_2$ to $\bar{v}$, which is a contradiction. □

**Observation 2** *There does not exist a tight planar packing of three trees.*

**Proof.** The sum of the number of edges for three trees with $n$ vertices is $3(n-1)$. But the number of edges of a simple plane graph of order $n$ is at most $3n-6$. □

Garcia et al. proved the following theorems in the same paper[1].

**Theorem 2** *Let $T$ be any tree with $n$ vertices which is different from a star. Then, there exists a tight planar packing of $T$ and a copy of $T$.*

**Theorem 3** *Let $T_1$ be any tree with $n$ vertices which is different from a star and let $T_2$ be a path of order $n$. Then, there exists a tight planar packing of $T_1$ and $T_2$.*

Also, we showed the following theorem.

**Theorem 4 (Enomoto et al. [5])** *Let $T_1$ be any tree with $n$ vertices which is different from a star. Let $T_2$ be a non-star graph which is obtained from a star by adding at most one vertex for each edge. (If it is obtained by adding a vertex for any edge, it is called a firework. See the lower right hand of Figure 2.) Then, there exists a tight planar packing of $T_1$ and $T_2$.*

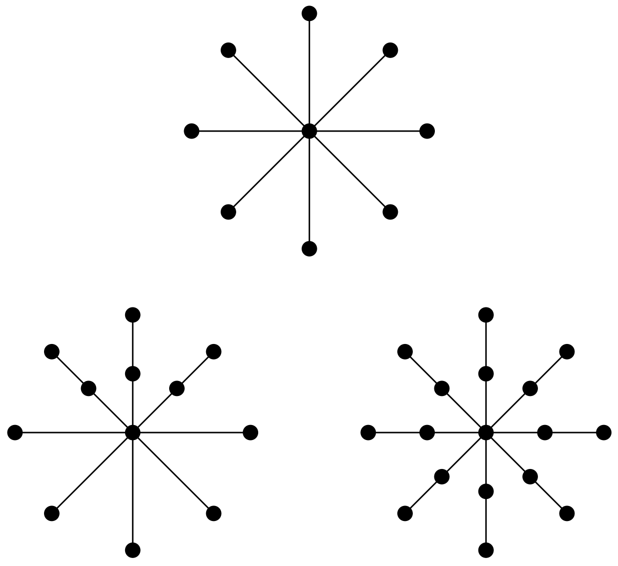


Figure 2: A star and graphs $T_2$ in Theorem 4

## 2 Main results

A caterpillar is a graph derived from a path by hanging any number of leaves from the vertices of the path.

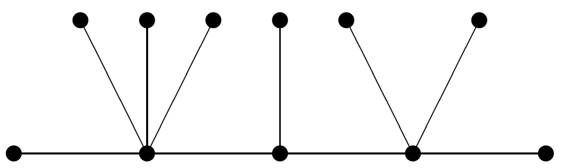


Figure 3: A caterpillar

The following is our main theorem.

**Theorem 5** *Let $T_1$ be any tree with $n$ vertices which is different from a star and let $T_2$ be any caterpillar with $n$ vertices which is different from a star. Then, there exists a tight planar packing of $T_1$ and $T_2$.*

We give a sketch of our proof. At first, we set $n$ vertices in some line $l$ and each vertex is labeled from 0 to $n-1$ in linear order. Next, we consider two embeddings of $T_1$ and $T_2$ into a plane with non-crossing edges. An embedding of $T_1$ is to be on the upper area of $l$ and an embedding of $T_2$ is to be on the lower area of $l$. Then, we should take care only that they do not share any edges. Here we give a key lemma as follows:

**Lemma 6** *Let $T_1$ be a caterpillar with $m$ vertices and $T_2$ be a tree with $n$ vertices. We assume that $m < n$. Take vertices $r_i \in V(T_i)$ for $i = 1, 2$, arbitrarily. Let $a$ be an integer such that $0 \leq a \leq n - m$. Then, there exist two bijection $\varphi : V(T_2) \to [0, n-1]$ and $\gamma : V(T_1) \to [a, a+m-1]$ satisfying:*

*(1) Each embedding of $T_1$ and $T_2$ according to $\varphi$ and $\gamma$ is non-self-intersecting.*

*(2) The embeddings of $T_1$ and $T_2$ according to $\varphi$ and $\gamma$ do not share any edge.*

*(3) The vertex $\varphi(r_2)$ is open in $\varphi(T_2)$ and $\gamma(r_1)$ is open in $\gamma(T_1)$.*

*(4) $\varphi(r_2) \neq \gamma(r_1)$*

*(5) $\varphi(r_2) \in \{0, n-1\} \cup [a+1, a+m-2]$*

We say that the vertex $\varphi(r_2)$ is *open* in $\varphi(T_2)$ if there does not exist any edge $e = (u, v)$ in $T_2$ such that $\varphi(u) < \varphi(r_2) < \varphi(v)$ or $\varphi(v) < \varphi(r_2) < \varphi(u)$. We define $\gamma(r_1)$ is *open* in $\gamma(T_1)$ similarly.

Theorem 5 is one of the special cases of Conjecture 1. Needless to say, our ultimate goal is to solve Conjecture 1. Toward the goal we need more general proof technique. We note that the above lemma is proved by using induction and the technique is new.

## References

[1] A. Garcia, C. Hernando, F. Hurtado, M. Noy and J. Tejel. Packing trees into planar graphs. *J. Graph Theory*, 40:172–181, 2002.

[2] M. R. Garey and D. S. Johnson. Computers and intractability - A guide to the theory of NP-completeness. W. H. Freeman and Co., 1979.

[3] S. M. Hedetniemi, S. T. Hedetniemi and P. J. Slater. A note on packing two trees into $K_n$. *Ars Combin.*, 11:149–153, 1981.

[4] M. Maheo, J. F. Saclé and M. Woźniak. Edge-disjoint placement of three trees. *European J. of Combinatorics*, 17:543–563, 1996.

[5] H. Enomoto, K. Kanda, T. Masui, Y. Oda, K. Ota. Private communications.

# A Topologically Robust Boolean Algorithm Using Approximate Arithmetic

Julian M. Smith [*] Neil A. Dodgson [†]

## Abstract

We present a previously unpublished, topologically robust algorithm for Boolean operations on polyhedral boundary models. The algorithm can be proved always to generate a result with valid connectivity if the input shape representations have valid connectivity, irrespective of the type of arithmetic used or the extent of numerical errors in the computations or input data. The main part of the algorithm, known as the basic Boolean algorithm, is based on a series of interdependent operations. The relationship between these operations ensures a consistency in the intermediate results that guarantees correct connectivity in the final result. Either a triangle mesh or polygon mesh can be used. The algorithm described can be extended naturally to the problem of computing the overlay of two cellular subdivisions, and also to operate in higher-dimensional domains.

## 1 Introduction

Problems of robustness are a major cause for concern in the implementation of computational geometry algorithms ([2, 5, 6, 7]). Most geometrical algorithms are a mix of numerical and combinatorial computations, and the approximate nature of the former often leads to logical decisions that are inconsistent, thus hindering the construction of a combinatorially correct result. In the context of boundary representations, inconsistent computations can lead to connectivity faults. There is also the problem that numerical errors can lead to the computed boundary intersecting itself.

A number of proposals have been put forward to address this problem. One approach favoured, particularly in the polyhedral domain, is to rely on exact arithmetic to avoid altogether the problems of numerical errors and inconsistencies [1, 4, 8]. Floating point filters are often used in conjunction to reduce the overhead of exact computations.

Certain methods take the topology-oriented approach. These are designed to guarantee a topologically or combinatorially correct result, irrespective of the extent of any numerical error in the computations or in the input data. As such they can be implemented using standard floating point arithmetic. Sugihara et al. [9] refer to such techniques for the Voronoi and Delaunay problems, the convex hull problem, and also for the intersection of convex polyhedra.

The algorithm presented here can be classed as topology-oriented. The main part of the algorithm, the *basic Boolean algorithm*, consists of a series of interdependent operations guaranteed to yield consistent intermediate results. This in turn ensures the final result has valid connectivity, provided both input structures have valid connectivity. The algorithm can be implemented using either a triangle or polygon mesh. For the triangle mesh variant of the basic algorithm there is a requirement to break up non-triangular facets into triangles.

The structure generated by the basic algorithm may have features that are redundant or are close to making the structure geometrically invalid. For that reason it is generally preferable to apply a data smoothing process to the structure to make it suitable for downstream operations.

The basic Boolean algorithm is described in section 2. Section 3 briefly discusses topological robustness and also the need for data smoothing. The process for triangulating facets is not covered.

## 2 The basic Boolean algorithm

The basic algorithm for the Boolean operation between two shapes $A$ and $B$ is performed as a series of interdependent operations. Each operation determines the relation between two entities, one from each shape, an *entity* being a vertex, edge (or half-edge), facet, or the entire shape. Hence there are 16 types of operation: one for each pairing of the four types of entity. Each operation type is considered as belonging to a particular level, 0 to 6, equal to the sum of the manifold dimensionalities of the two entities, $o_A$ and $o_B$. Each operation has a similar pattern: the result is influenced by the results of those operations one level lower, concerning (1) each boundary component of $o_A$ in turn, and $o_B$; and (2) $o_A$, and in turn each boundary component of $o_B$. This leads to a dependency hierarchy between the types of operation, as shown in figure 1. Operations at level 0-3 determine the point at which the entities intersect, while those at level 3-6 work towards constructing the result.

The operations at level 3 take a pivotal role between

[*]Rainbow Group, Computer Laboratory, University of Cambridge, jms222@cl.cam.ac.uk

[†]Rainbow Group, Computer Laboratory, University of Cambridge, nad@cl.cam.ac.uk

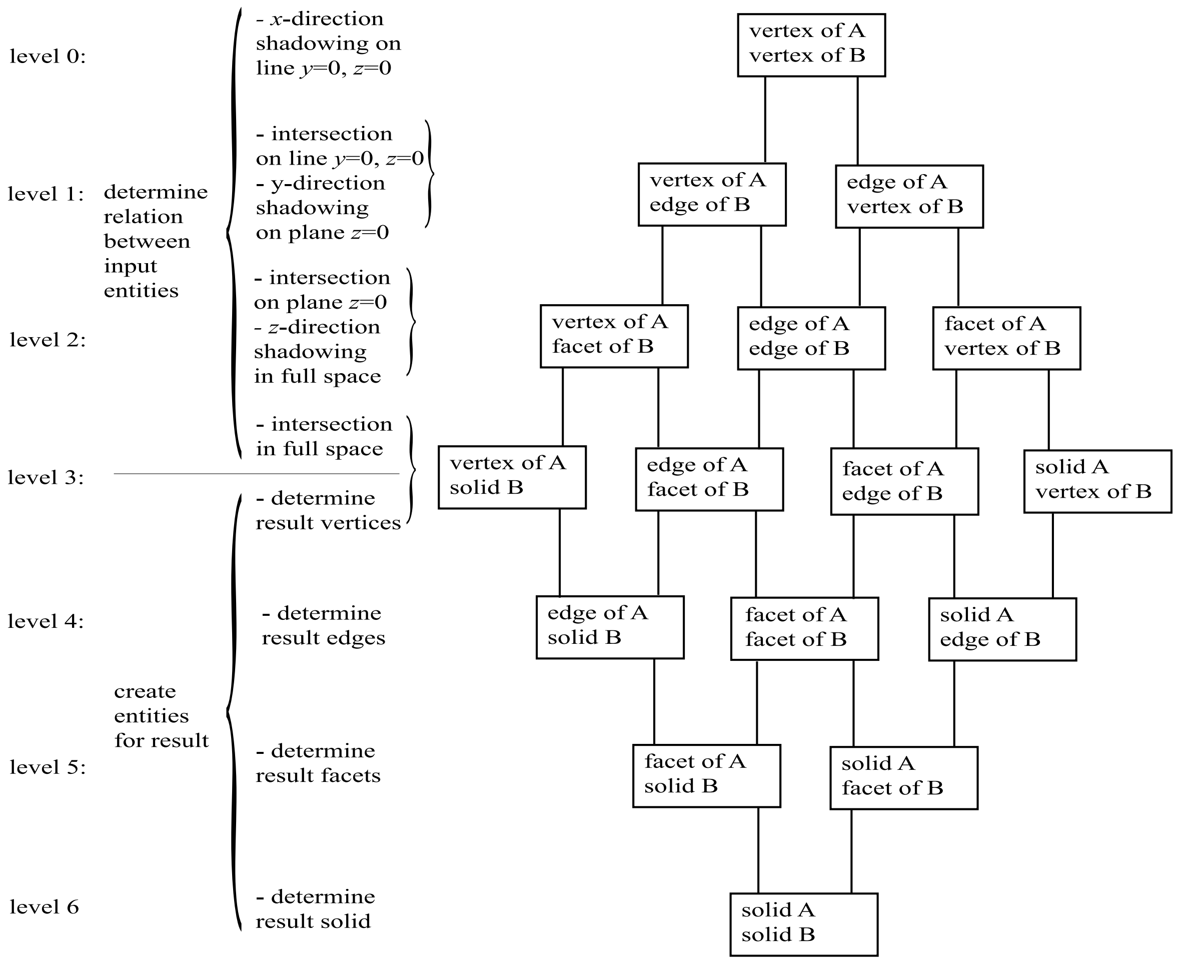


Figure 1: The hierarchy of operations for the basic Boolean algorithm.

the two stages. They determine whether a vertex of one solid lies inside or outside the other solid, and whether an edge of one solid intersects a facet of the other solid. The latter operation also determines the point of intersection. All operations assume object $B$ to be perturbed by an infinitesimal distance in each of the three axis directions. This resolves the problem of degeneracy, in the manner described by Edelsbrunner and Mücke [3] in the context of exact computations. Thus a point is deemed to lie inside or outside a solid, never on the boundary; likewise, an edge either intersects a facet or it does not, they are never deemed simply to touch. This makes special case handling unnecessary.

An *intersection function*, $X_{ij}(o_A, o_B)$, ascertains whether entities $o_A$ and $o_B$ intersect, $i$ and $j$ indicating the manifold dimensionality of each entity. A non-zero function value indicates that the entities intersect; the sign of the function value indicates the nature of the intersection, such as whether the edge enters or exits the solid through the facet. Intersection functions are used at lower levels too, where $i{+}j{<}3$. In general terms, $X_{ij}(o_A, o_B)$ operates in $(i{+}j)$-dimensional space or subspace, as specified by the first $i{+}j$ coordinate values of the Cartesian representation used in the computation. Hence level 3 intersection functions ($i{+}j{=}3$) operate in full 3D space. Level 2 intersection functions ($i{+}j{=}2$) operate in effect in the $(x, y)$ plane, so $z$ coordinate values are ignored when determining if the entities intersect. However if they do intersect, the $z$-values of the intersection point are determined for the use in subsequent calculations. Two such values are required: one for each entity. In turn, level 1 intersection functions operate simply on the $(x)$ line, and two sets of $y$ and $z$ values are determined at any intersection

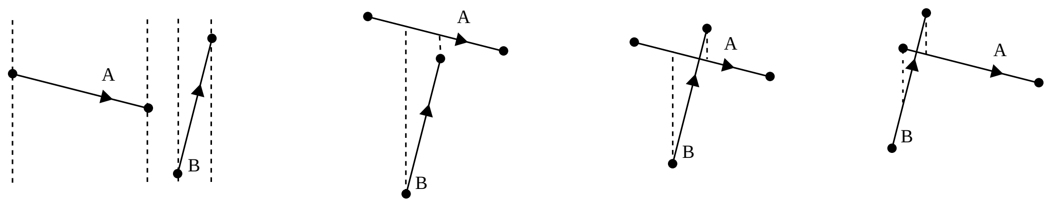


Figure 2: Demonstration of how intersection between two edges in $(x, y)$ space is determined. The two edges intersect only if they overlap in $(x)$ space ($\max x_B \geq \min x_A$ and $\min x_B < \max x_A$) and if $y_B \geq y_A$ at one end of the overlap range in $(x)$ and $y_B < y_A$ at the other (i.e. at $x = \max(\min x_A, \min x_B)$ and $x = \min(\max x_A, \max x_B)$).

point. One can also consider there to be a level 0 'intersection function', $X_{00}(v_A, v_B)$, that operates at the origin point; it always takes the value 1, with the 'intersection point' located at $v_A$ and $v_B$.

The calculations involve *shadow functions*, $S_{ij}(o_A, o_B)$, that operate in $(i+j+1)$-dimensional space or subspace as defined by the first $i+j+1$ coordinate values. If $X_{ij}(o_A, o_B) = 0$ then $S_{ij}(o_A, o_B) = 0$. If $X_{ij}(o_A, o_B) \neq 0$, the value is determined by considering the next coordinate value at the point of intersection: $\xi_A$ on $o_A$ and $\xi_B$ on $o_B$, where $\xi$ is $x$ if $i+j$=0, $y$ if $i+j$=1, or $z$ if $i+j$=2. $S_{ij}(o_A, o_B) = X_{ij}(o_A, o_B)$ if $\xi_B \geq \xi_A$, or 0 otherwise.

The intersection function $X_{ij}(o_A, o_B)$ is evaluated as the sum, with appropriate $+/-$ signs, of each of the shadow function values: $S_{i-1,j}(c, o_B)$ for each boundary component $c$ of $o_A$ (if $i > 0$); and $S_{i,j-1}(o_A, c)$ for each boundary component $c$ of $o_B$ (if $j > 0$). The individual formulae are listed in table 1.

The intersection point associated with a non-zero value of $X_{ij}(o_A, o_B)$ is determined by considering two lower-level intersection points associated with a non-zero value of $X_{i-1,j}$ or $X_{i,j-1}$, one for which $\xi_B \geq \xi_A$ and one for which $\xi_B < \xi_A$. The intersection point for $X_{ij}$ is obtained by linearly interpolating the two lower-level intersection points so that $\xi_B = \xi_A$. Figure 2 demonstrates this for the case $X_{11}(e_A, e_B)$.

Following the determination of the level 3 intersection points the structure representing the result is constructed. The result contains two types of vertices: retained vertices, which are copies of original vertices that lie on the appropriate side of the other solid (outside for the union operation, inside for the intersection operation), and intersection vertices, which are new vertices located at each point where an edge intersects a facet.

The operations at level 4 construct the edges: retained edges (and part-edges), and intersection edges (between two facets). For each operation, a series of start-vertices and end-vertices is obtained from the level 3 operations, and these are paired up to form edges in the new structure.

Retained facets (or part-facets) are determined at level 5; these are bordered by half-edges computed at level 4. For the triangle mesh variant of the algorithm, any non-triangular facet must be broken up into triangles. For the polygon mesh variant, the polygonal region is retained as it is.

Finally, at level 6 the resulting solid is generated from the retained facets.

## 3 Topological robustness, numerical accuracy, and implementation

Topological robustness of the basic algorithm is assured by the formulae used, assuming $A$ and $B$ both have valid connectivity. For the calculations at levels 1, 2 and 3, it can be proved that if $X_{ij} \neq 0$, then there will be at least one lower-level intersection where $\xi_B \geq \xi_A$ and also one where $\xi_B < \xi_A$, hence it will always be possible to interpolate to determine the point of intersection. At level 4, it can be proved that the number of start-vertices will always equal the number of end-vertices, so it will always be possible to determine segments from which to construct edges. At level 5, it can be shown that every end-vertex of a half-edge of a particular facet is also the start-vertex of a half-edge of the same facet, and at level 6, every half-edge going from vertex $v$ to $w$ is matched by a half-edge going from vertex $w$ to $v$. Hence the structure generated satisfies the constraints that ensure it has correct connectivity. Proofs of these statements will be published separately.

Geometrical validity of the result is *not* assured by the basic Boolean algorithm. The structure generated can have artifacts that can be considered unsuitable: coincident vertices, coincident and opposing facets, coincident and opposing half-edges within a facet, zero-length edges, and zero-area facets. Coincident and opposing facets or half-edges make the structure on the borderline of being geometrically invalid; it may even turn out invalid due to numerical errors in computing the result, or else it may become invalid after applying a subsequent operation such as a repositioning translation. It is therefore appropriate to process the structure to remove the artifacts before passing on the data structure to any process likely to be adversely affected by geometrical errors. This data

| *level* | *formula* | | *situation leading to value of* $+1$ |
|---|---|---|---|
| 0 | $X_{00}(v_A, v_B) =$ | $1$ | |
| 1 | $X_{01}(v_A, e_B) =$ | $S_{00}(v_A, v_e(e_B)) - S_{00}(v_A, v_s(e_B))$ | $v_A$ lies within $e_B$, with $e_B$ going left to right |
| | $X_{10}(e_A, v_B) =$ | $-S_{00}(v_e(e_A), v_B) + S_{00}(v_s(e_A), v_B)$ | $v_B$ lies within $e_A$, with $e_A$ going left to right |
| 2 | $X_{02}(v_A, f_B) =$ | $-\sum_{h \in \partial f_B} S_{01}(v_A, h)$ | $v_A$ lies within $f_B$, with $f_B$ going anti-clockwise (denoting that it faces upwards) |
| | $X_{11}(e_A, e_B) =$ | $S_{01}(v_e(e_A), e_B) - S_{01}(v_s(e_A), e_B)$ $+S_{10}(e_A, v_e(e_B)) - S_{10}(e_A, v_s(e_B))$ | $e_A$ crosses $e_B$ from left to right |
| | $X_{20}(f_A, v_B) =$ | $\sum_{h \in \partial f_A} S_{10}(h, v_B)$ | $v_B$ lies within $f_A$, with $f_A$ going anti-clockwise |
| 3 | $X_{03}(v_A, B) =$ | $\sum_{f \in \partial B} S_{02}(v_A, f)$ | $v_A$ lies within $B$, with faces facing outwards |
| | $X_{12}(e_A, f_B) =$ | $-S_{02}(v_e(e_A), f_B) + S_{02}(v_s(e_A), f_B)$ $-\sum_{h \in \partial f_B} S_{11}(e_A, h)$ | $e_A$ crosses $f_B$ going to the outer side |
| | $X_{21}(f_A, e_B) =$ | $-\sum_{h \in \partial f_A} S_{11}(h, e_B)$ $+S_{20}(f_A, v_e(e_B)) - S_{20}(f_A, v_s(e_B))$ | $e_B$ crosses $f_A$ going to the outer side |
| | $X_{30}(A, v_B) =$ | $-\sum_{f \in \partial A} S_{20}(f, v_B)$ | $v_B$ lies within $A$, with faces facing outwards |

Table 1: Formulae for intersection functions. $\partial f$ denotes the collection of half-edges that border facet $f$; $v_s(e)$ and $v_e(e)$ denote the start- and end-vertex of edge (or half-edge) $e$.

smoothing process consists of a number of Euler-type operations (as described in [5]), simplifying the structure while preserving topological correctness, and restricting any change to the shape boundary (which may be necessary) to within specified bounds.

The first author devised and successfully implemented the Boolean algorithm for use within a widely circulated commercial CAD product, for which Boolean operations are used principally to convert CSG models of industrial plant components to polyhedral approximations. The product and algorithm continue to be used extensively several years after release. The general polygonal mesh variant of the algorithm was adopted, since it was found to be more efficient than an initial trial version based on the triangular mesh. It was found to be more efficient to apply the data smoothing process once at the end of a sequence of basic Boolean operations rather than after each individual operation. The data smoothing process that was implemented does not have a full theoretical backing, so it is not provably fully robust in performing the task required of it, but it has turned out to be sufficiently reliable. Users of the released product only ever reported one fault that turned out to be related to the data smoothing process: a problem of non-termination, which was resolved by enforced early termination.

## References

[1] CGAL. http://www.cgal.org/, 2006.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry.* Springer, 1997.

[3] H. Edelsbrunner and E. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1), January 1990.

[4] S. Fortune. Polyhedral modelling with exact arithmetic. In *Proc. 3rd Symp. Solid Modeling*, pages 225–234. ACM Press, NY, 1995.

[5] C. Hoffmann. *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann Publishers, Inc., 1989.

[6] C. Hoffmann. Robustness in geometric computations. *Journal of Computing and Information Science in Engineering*, 1:143–156, 2001.

[7] C. Hoffmann, J. Hopcroft, and M. Karasik. Robust set operations on polyhedral solids. *IEEE Computer Graphics & Applications*, 9(6):50–59, 1989.

[8] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[9] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation—an approach to robust geometric algorithms. *Algorithmica*, 27:5–20, 2000.

# A Small Improvement in the Walking Algorithm for Point Location in a Triangulation

Ivana Kolingerová*

## Abstract

The paper shows a simple technique which saves some edge tests in the walking algorithm for point location. The walking technique does not achieve the logarithmic per point complexity of the location-data-structure-based methods but does not need any auxiliary data structure and is very simple to implement, therefore, it is very popular in practice. The suggested idea did not bring a substantial improvement in our tests but it is very simple and there is an open door to further more substantial improvement in the future research.

## 1 Introduction

Point location in a triangulation is a very frequent task. Most effective solutions use hierarchical data structures, such as a DAG [1], [5], a skip list [10], a quadtree, buckets [9], a data structure with a random sampling [7], [2] or a uniform grid [8], [11]. These structures are very effective and bring a complexity $O(\log\ n)$ per point where $n$ is the total number of points in the triangulation. However, their disadvantage is a memory consumption, which, although linear in $E^2$, still can bring a substantial limitation to the program usefulness as the data sets processed today are very huge. Also, implementation effort for most of these data structures may be nontrivial. Therefore, practical programmers turn very often to a 'pragmatic' solution, represented by the walking type of location algorithm, where the point is located by traversing from one triangle to another according to some kind of test of the point position against the triangle boundaries. Such an approach is less effective, bringing $O(n^{1/3})$ up to $O(n^{1/2})$ per one point location but no extra location data structures are needed and so no extra memory is consumed [4], [3], [6].

There are several walking algorithms, differing in the strategy how to find the next triangle from the current one, the most effective one seems to be the so-called remembering stochastic walk. This paper shows a very simple improvement of this walking algorithm that may save about 8% of edge tests. Theoretical improvement could be up to 50% but we have not been able to achieve this efficiency yet.

Section 2 explains the remembering stochastic walk and the suggested improvement. Section 3 explains experiments and results and section 4 concludes the paper.

## 2 Remembering stochastic walk and the suggested improvement

Walking strategy generally means that from some starting triangle, we inspect the triangles one after another, traveling over the edge into that triangle neighbour which looks the best choice to approach the triangle containing the query point $q$. The starting triangle can be chosen in random, or it is the triangle visited most recently, or by brute force choice from a random subset of triangles according to their distances from the located point. The walking can traverse all the triangles intersected by the line segment originating at a vertex of the starting triangle and ending at the query point - this is the so-called *straight walk*. Or the transfer can be divided into 2 axes, approaching first in one and then in the other axis, so-called *orthogonal walk*. The *visibility walk* uses an orientation test of the triangle edge and the query point, a bad sign of the orientation test reveals which edge to cross to continue the search. As the visibility walk can cycle for a non-Delaunay triangulation, it can be more properly implemented in a randomized version as the stochastic walk - randomization means here that the choice of the first triangle edge to be tested is random, which prevents an infinite loop. The last modification is the *remembering stochastic walk* which remembers over which edge we came into the triangle. It does not test this edge because we already know the result of this test, after all. All these walking strategies and their comparison in $E^2$ and $E^3$ can be found in [3].

The algorithm of remembering stochastic walk is given in Alg.1 (adapted from [3]). For further acceleration, it is good to combine the walking algorithm with some preprocessing where a randomly selected set of triangle vertices from the triangulation is tested on distance to the query point and the smallest distance from a vertex to the query point chooses the starting triangle [6]. A proper size of this set is $O(n^{1/3})$ triangle vertices where $n$ is the total number

*Department of Computer Science and Engineering, University of West Bohemia, Pilsen, Czech Republic, kolinger@kiv.zcu.cz

of vertices in the triangulation. Although the distance tests are expensive, still this technique brings improvement both in the number of edge tests and in the total runtime.

```
Remembering stochastic walk ($t$,$q$)
// traverses the triangulation T
// from the triangle $t$ to the query point $q$
// using the remembering stochastic walk
begin
  $previous$ := $t$ ; found := false;
  while not found do
  begin
    $e$ := random edge from $t$;
    $p$ := the vertex of $t$ not contained in $e$;
    $nb$ := neighbour ($t$ over $e$);
    if ($nb$ is not equal to $previous$) and
      ($q$ on the other side of $e$ than $p$) then
      begin
        $previous$ := $t$; $t$ := $nb$
      end
    else
      begin
        $e$ := next edge of $t$;
        $p$ := the vertex of $t$ not contained in $e$;
        $nb$ := neighbour ($t$ over $e$);
        if ($nb$ is not equal to $previous$) and
          ($q$ on the other side of $e$ than $p$) then
          begin
            $previous$ := $t$; $t$:=$nb$
          end
        else
          begin
            $e$ := next edge of $t$;
            $p$ := the vertex of $t$ not contained in $e$;
            $nb$ := neighbour ($t$ over $e$);
            if ($nb$ is not equal to $previous$) and
            ($q$ on the other side of $e$ than $p$) then
              begin
                $previous$ := $t$;
                $t$ := $nb$
              end
            else found := true
          end
      end
  end
end
// now $t$ contains $q$
```

Remembering stochastic walk

Algorithm 1

An average number of edge tests per triangle is 1.5 because in each triangle (with the exception of the first one), we either find a bad orientation at the first attempt and go to the triangle sharing this edge, or we have to test one more edge and the orientation test either sends us further, or acknowledges that the query point is inside this triangle. The starting triangle may need one more test but this is not important for the average value.

An improvement of this algorithm is straightforward: if we want to be better, we should test only one edge per triangle. Is it possible? With the exception of the first and the last triangle, yes: we know the result of test of the edge which led us to the current triangle. If we test one more edge, we can either get the result 'go to the neighbouring triangle sharing this edge', or the result 'do not go over this edge, you should stay inside or go to the triangle over the third triangle edge'. We cannot decide for sure between the 2 latter possibilities without one more test, but the key point is that the answer 'stay inside' is valid for the last triangle only and is highly improbable in comparison with the answer 'leave the triangle over the non-tested edge', as we usually test many triangles before we come to the triangle containing the query point. In this way, we can save as much as one edge test per triangle.

Black point of this improvement is that with one edge test per triangle, we are not able to recognize that we are already in the triangle containing the query point, and we could continue in the walking for ever without recognizing the end. Therefore, we can use the 'fast walking' strategy only for some number of steps which we expect it is usually necessary to approach the goal triangle, and then continue more slowly by the remembering stochastic walk, until we find the goal triangle.

This combined strategy would work perfectly if we knew exactly how many steps the algorithm will need to find the goal triangle but this is not the case. According to [6] and our experiments, the number of visited triangles is $O(n^{1/3})$ in average, so we can expect about this number of steps to get into the goal triangle. Bad news is that dispersion of the number of visited triangles per query point is very high, therefore, in many cases we stop the faster search too early and in many cases too late, which increases the total number of the tests. The whole algorithm which we called *fast walk* is given as Alg.2 and results can be seen in the next section.

```
**Fast walk** ($t$,$q$)
// traverses the triangulation T
// from the triangle $t$ to the query point $q$
// using fast walk
// *nsteps* is the number of steps for the fast walk,
// usually $O(n^{1/3})$
// where $n$ is the number of vertices in the triangulation
begin
  previous:= t;
  for i := 1 to nsteps do
    begin
      $e$ := random edge from $t$;
      $nb$ := neighbour ($t$ over $e$);
      while $nb$ is equal to previous do
        begin
          $e$ := next edge of $t$; $nb$:= neighbour ($t$ over $e$);
        end ;
      $p$ := the vertex of $t$ not contained in $e$;
      if ($q$ on the same side of $e$ as $p$) then
        begin
          $e$ := next edge of $t$; $nb$ := neighbour ($t$ over $e$)
        end;
      previous:= $t$ ; $t$ :=$nb$
    end
end
// we do not know whether $t$ contains $q$,
// so we must continue by the remembering
// stochastic walk
Remembering stochastic walk ($t$, $q$);
// now $t$ contains $q$
```

Fast walk

Algorithm 2

## 3 Experiments and results

We tested the program as a part of an incremental insertion algorithm for the Delaunay triangulation, programmed in Delphi, under MS Windows operating system. In order to have results independent on the platform, we measured the number of edge tests necessary to construct the triangulation for up to 5 million of points. We measured uniformly distributed points, but also points in clusters and other types of data. We show the uniform data as they provided the average values. The suggested improvement is compared to the remembering stochastic walk. Both algorithms were programmed with the use of preprocessing according to [6] as the version with preprocessing is quicker and it tests fewer edges in all cases. Results for the straight walk are not given as they were worse than those of the remembering walk in all cases.

Figure 1 shows how many triangle tests are necessary in the remembering stochastic walk for each query point within the triangulation process. The average number of visited triangles can be approximated by $2n^{1/3}$ to $2.15n^{1/3}$ for $n = 1000$ up to 5 million but the dispersion is high, see an example in Fig.1. Due to this dispersion, we used less fast walk steps than would correspond to this function. (We should stress that in the application for triangulation construction, $n$ is different for each inserted point and is equal to the number of points already inserted into the triangulation, not to the total number of points to be inserted.)

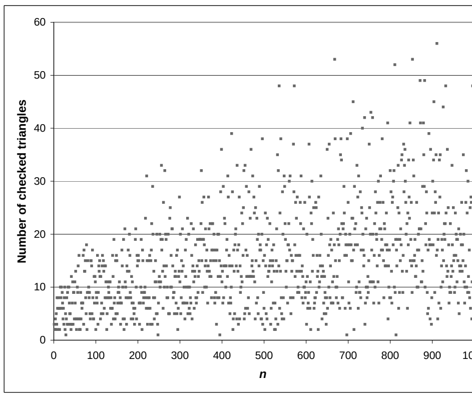


Figure 1: The number of checked triangles in the remembering walk during construction of the Delaunay triangulation on 1000 points.

Table 1 shows comparison between the number of tested edges in the remembering stochastic walk and in the fast walk. We tested various functions derived from $O(n^{1/3})$ for derivation of the number of steps for fast walk (*nsteps* in Alg.2). The best results of the fast walk presented in this table were obtained by the number of fast walk steps equal to $1.15n^{1/3}$. Improvement is much lower than expected, only about 8% of the edge tests. The reason for this smaller success than hoped is the already mentioned dispersion in the number of traversed edges: the expected number of triangles that can be fast walked varies so much that a decrease in the number of tested edges is nearly compensated by inspecting some triangles in vain because the goal triangle was not recognized in time; in some cases the situation is the opposite, fast walk is stopped too early. Unfortunately, there is no way how to find the correct number of tests for the given case in advance. Maybe there exists some simple, elegant and brilliant idea how to get closer to the theoretical bound 50% improvement but we have not come to it yet.

## 4 Conclusion

The paper presents a fast walk, a simple modification of the remembering stochastic walk algorithm for locating a point in a triangulation. The method is very simple and does not consume any extra memory for data structure, improvement achieved in the tests is not significant at present but the simplicity of the idea may inspire further research to achieve a theoretically possible 50% improvement against the original remembering walk algorithm.

| $n$ | Rem.st.wlak | Fast walk | Savings |
|---|---|---|---|
| $10^4$ | 469 462 | 445 097 | 5.47 |
| $5.10^4$ | 4 028 040 | 3 759 661 | 7.14 |
| $10^5$ | 10 084 883 | 9 372 859 | 7.60 |
| $5.10^5$ | 85 496 066 | 79 068 466 | 8.13 |
| $10^6$ | 214 296 506 | 197 641 298 | 8.43 |
| $5.10^6$ | 1 816 818 470 | 1 670 737 631 | 8.74 |

Table 1: Number of edge tests for the remembering stochastic walk and for the fast walk

## 5 Acknowledgement

The author would like to thank to ing. J.Kohout for inspiration to inquire into the walking algorithms.

## References

[1] M. de Berg, M. van Kreveld, M. Overmars and 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 and M. Teillaud. Walking in a triangulation. *Proceedings of the 17th Annual Symposium on Computational Geometry 2001*, 106-114, 2001.

[4] L.J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans Graphics*, 4(2):75-123, 1985.

[5] I. Kolingerová and B. Žalik. Improvements to randomized incremental Delaunay insertion. *Computers & Graphics*, 26, 477-490, 2002.

[6] E.P. Mücke, I. Saias and 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.

[7] K. Mulmuley. Randomized multidimensional search trees: dynamic sampling. *Proceedings of the 7th Annual Symposium on Computational Geometry 1991*, 121-131, 1991.

[8] S.W. Sloan A fast algorithm for constructing Delaunay triangulations in the plane. *Adv Eng Software*, 9(1):34-55, 1987.

[9] P. Su and R.L.S. Drysdale. A comparison of sequential Delaunay triangulation algorithms. *Proceedings of the 11th Annual Symposium on Computational Geometry 1995*, 61-70, 1995.

[10] M. Zadravec and B. Žalik. An almost distribution-independent incremental Delaunay triangulation algorithm. *The Visual Computer,* 21(6):384-396, 2005.

[11] B. Žalik and I. Kolingerová. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *Int.J. Geographical Information Science*, 17(2):119-138, 2003.

# A certified algorithm for the InCircle predicate among ellipses

Ioannis Z. Emiris[*] Elias P. Tsigaridas[*] George M. Tzoumas[*]

## Abstract

This paper examines the InCircle predicate among ellipses in the Euclidean plane, under the exact computation paradigm. The ellipses are non-intersecting and given in parametric representation. We present a subdivision-based algorithm and implement it in Maple and CORE.

## 1 Introduction

We study the InCircle predicate for the Voronoi diagram of ellipses. This is the hardest predicate for implementing the algorithm of [7] and has not been solved in the exact computation paradigm. The work coming closest to ours is [5]: The authors essentially trace the bisectors in order to compute the Voronoi cells of arbitrary curves up to machine precision. Their algorithm uses floating point arithmetic; they claim that their software works well in practice. Although they argue that their algorithm can be extended to exact arithmetic, they do not explain how. For instance, they do not discuss degenerate configurations. Our implementations are exact but can also run with any prescribed precision.

It seems hard, for the algebraic approach of [3], to yield a fast solution. All four predicates of the incremental algorithm [7] were studied in [4], including a certified subdivision-based algorithm for InCircle, implemented in Maple. In this paper, we offer a better implementtion using the CORE library [6]. The algorithm "moves" on the border of parametrically defined ellipses. This avoids computing the Voronoi circle explicitly.

The Voronoi circle is specified at any desired accuracy. This is achieved by refining the interval expressing its 3 tangency points until the predicate can be decided; in fact, all tangency points are expressed as a function of one of them. Exactness is guaranteed by root separation bounds.

Let the length of the axes be $2\alpha, 2\beta$ and $(x_c, y_c)$ be the center. We use the parametric representation:

$$\begin{aligned} x(t) &= x_c - (\alpha(1-w^2)t^2 + 4\beta w t - \alpha(1-w^2))/d \\ y(t) &= y_c + 2(-\alpha w t^2 + \beta(1-w^2)t + \alpha w)/d, \end{aligned}$$

[*]Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece, {emiris,et,geotz}@di.uoa.gr

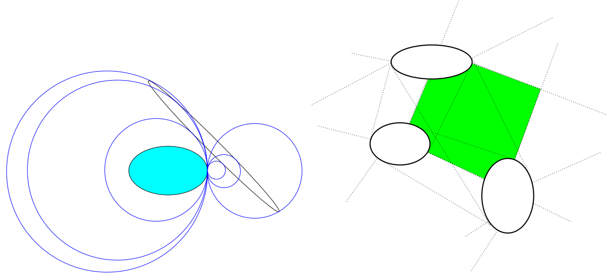


Figure 1: Left: The 6 bitangent circles. The Apollonius circle is the $4^{th}$ from the left. Right: Starting intervals for $t, r, s$ (and region of the Voronoi vertex)

where $d = (1+w^2)(1+t^2)$, $t = \tan(\theta/2) \in (-\infty, \infty)$, $\theta$ is the angle that traces the ellipse, $w = \tan(\frac{\phi}{2})$, and $\phi$ is the rotation angle between the major and horizontal axes. We denote by $E_t$ an ellipse parameterized by $t$.

## 2 The bitangent circle

**Lemma 1** *Given 2 ellipses and a point on the first, there may exist up to 6 real bitangent circles, tangent at the specific point. This bound is tight. Only one such circle is* external *to both ellipses.*

We call this unique *external* bitangent circle the *Apollonius* circle of the 2 ellipses, e.g. the third circle from the right in fig. 1 (left).

Given ellipses $E_t, E_r$, the tangency points of any Apollonius circle lie inside their Convex Hull (CH). This offers a starting point to begin our search for the tangency point of the Voronoi circle within a *continuous* range on the boundary of an ellipse. Now, consider all bitangent circles to $E_t, E_r$, tangent at point $t$ of $E_t$.

We shall compute arc $(r_1, r_2)$ on $E_r$ which contains *only* the tangency point of the Apollonius circle, *isolating* it from the tangency points of non-external bitangent circles. Consider all bitangent circles at $t$. Also, consider the lines from $t$ tangent to $E_r$ at points $r_1, r_2$. They define two arcs on $E_r$. Arc $(r_1, r_2)$, whose interior points lie on the same side of line $r_1 r_2$ as $t$, is called a *visible arc*.

Visible arc $(r_1, r_2)$ contains only tangency points of bitangent circles at $t$, which are externally tangent to $E_r$, but may be internally tangent to $E_t$. They include the Apollonius circle of $E_t, E_r$, tangent at $t \in E_t$.

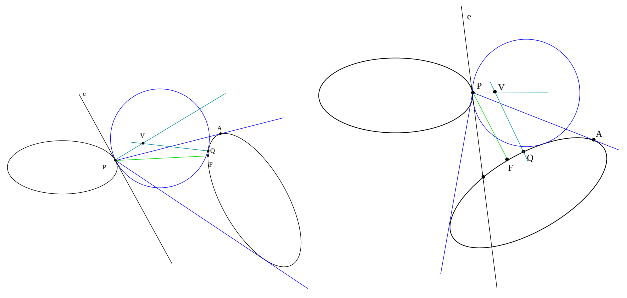


Figure 2: The visible arc and the Apollonius circle

**Lemma 2** *Given is a point $P = (x(t), y(t))$ on $E_t$. Consider line $\epsilon$, tangent at $P$ (cf. fig. 2). If $\epsilon$ does not intersect $E_r$, then the visible arc contains a unique Apollonius circle. Otherwise, the endpoints of such an arc are: the intersection of $\epsilon$ with $E_r$ and the endpoint of the visible arc which lies on the opposite side of $E_t$ with respect to $\epsilon$.*

Given ellipses $E_t, E_r$ their bisector $B(t,r)$ is a bivariate polynomial of degree 6 in $t$ and 6 in $r$. The above lemma provides an isolating interval for the unique root $\hat{r}$ of $B(t,r)$, which lies on the visible arc of $E_r$ with respect to some fixed $t$. In other words, $\hat{r}$ corresponds to the Apollonius circle.

Given a point $(x(t), y(t))$ on $E_t$, the *squared radius* of the Apollonius circle of $E_t, E_r$ tangent to $E_t$ at that point is denoted by $f_{tr}(t)$. It follows that

$$f_{tr}(t) := \big(v_1(t,\hat{r}) - x(t)\big)^2 + \big(v_2(t,\hat{r}) - y(t)\big)^2,$$

where $\hat{r}$ is the root of the bisector that corresponds to the Apollonius circle, when we fix $t$, and $(v_1, v_2)$ is the intersection of the normals at $t$ and $\hat{r}$.

In the sequel, we assume that $f_{tr}(t)$ is defined on a continuous interval $(a, b)$. The interval can also be of the form $(-\infty, a)\cup(b, \infty)$, but in this case the problem is identical or easier.

**Lemma 3** *Function $f_{tr}(t)$ consists of two strictly monotone parts, one decreasing and one increasing.*

We have not proved the function's convexity, though this is implied by numerical examples.

Our overall algorithm maintains an interval that contains the tangency on $E_t$. At every iteration, it picks some value for $t$ within the interval and solves $B(t,r)$ for finding $\hat{r}$, which was defined above. These values are used to determine the sign of $\mathcal{S}$, to be introduced below.

## 3 Deciding the predicate

The Voronoi circle is the circle which is externally bitangent to $E_t, E_r$, and $E_t, E_s$ at the same time. Its tangency point on $E_t$ is defined by the condition:

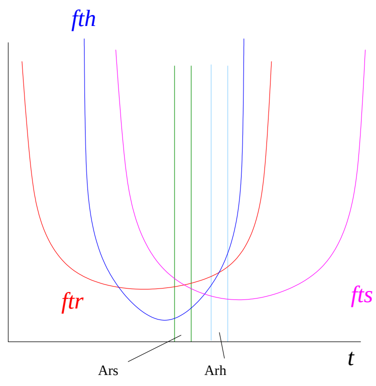


Figure 3: Deciding the predicate

$$\mathcal{S}_{trs}(t) = 0, \text{ where } \mathcal{S}_{trs}(t) = f_{tr}(t) - f_{ts}(t),$$

where $f_{tr}$ (and similarly $f_{ts}$) in the case of ellipses becomes

$$f_{tr}(t) = \frac{1}{4} P_t(t) \left( \frac{A_{tr}(t,\hat{r})}{(1+t^2)(1+\hat{r}^2) D_{tr}(t,\hat{r})} \right)^2.$$

In the above equation, $P_t(t)$ has no real roots, $A_{tr}$ is a bivariate polynomial of degree 2 in $t$ and 4 in $r$ and $D_{tr} \neq 0$, unless the normals at $t, \hat{r}$ are parallel.

We factor $\mathcal{S}_{trs}(t)$ as follows:

$$\frac{P_t(t) \cdot [Q_1(t,\hat{r},\hat{s}) - Q_2(t,\hat{r},\hat{s})] \cdot [Q_1(t,\hat{r},\hat{s}) + Q_2(t,\hat{r},\hat{s})]}{4\big[(1+t^2)(1+\hat{r}^2)(1+\hat{s}^2) D_{tr}(t,\hat{r}) D_{ts}(t,\hat{s})\big]^2} \quad (1)$$

We use a customized bisection to approximate a root of $\mathcal{S}_{trs}(t)$. We only need to determine the sign $Q_1 - Q_2$ and $Q_1 + Q_2$, since the rest of the terms in (1) are always positive. This way we avoid computing $f$ explicitly.

We express the Voronoi circle of $E_t, E_r, E_s$ by an interval containing $t$, such that $(x(t), y(t))$ is the tangency point on $E_t$.[1] We start by the initial interval $[a, b]$ that contains the tangency point and subdivide it by bisection. The subdivision operator yields

$$\begin{array}{ll} [\frac{a+b}{2}, \frac{a+b}{2}], & \text{if } \mathcal{S}_{trs}(\frac{a+b}{2}) = 0, \\ [a, \frac{a+b}{2}], & \text{if } \mathcal{S}_{trs}(a)\mathcal{S}_{trs}(\frac{a+b}{2}) < 0, \\ [\frac{a+b}{2}, b], & \text{otherwise.} \end{array}$$

**Theorem 4** *Let $x \in [a,b]$ be the root of $\mathcal{S}_{trs}(x)$. If $\mathcal{S}_{trh}(x) > 0$, then $E_h$ intersects the Voronoi circle of $E_t, E_r, E_s$. If $\mathcal{S}_{trh}(x) < 0$, then $E_h$ lies outside the Voronoi circle. Otherwise, $E_h$ is externally tangent to this circle.*

[1]This interval might contain tangency points of other non-external tritangent circles, but they don't interfere with our approach, since it deals only with externally bitangent circles.

Note that there is no such case such as internal tangency. This is due to the fact that we deal only with externally tangent circles.

Clearly, there is a neighborhood $\mathcal{U}$ of $x$ where $\operatorname{sgn}(S_{trh}(u)) = \operatorname{sgn}(S_{trh}(x))$, $\forall u \in \mathcal{U}$. In our implementation, to find $\mathcal{U}$, it will suffice to separate the roots of $\mathcal{S}_{trs}, \mathcal{S}_{trh}$.

We now establish the *exactness* of our algorithm. Consider system $\Delta_1(v_1, v_2, q) = \Delta_2(v_1, v_2, q) = \Delta_3(v_1, v_2, q) = q - v_1^2 - v_2^2 + s = 0$ [4], where $(v_1, v_2)$ is the center and $s$ the squared radius of the Voronoi circle. Let us eliminate $v_1, v_2, q$; the resultant $R(s)$ is of degree 184 in $s$ and has coefficient bit size $3 \cdot 56 \cdot \tau_\Delta = 168\tau_\Delta$. Here 56 equals the mixed volume of the system $\Delta_i, \Delta_j, q - v_1^2 - v_2^2 + s$, if we consider $s$ as a parameter, and $\tau_\Delta$ denotes the bit size of the coefficients of $\Delta_i$, where $1 \leq i, j \leq 3$ and $i \neq j$.

The minimum distance between two real roots of a polynomial $P$ of degree $d$ and bit size $\tau$ is $sep(P) \geq d^{-(d+2)/2}(d+1)^{(1-d)/2}2^{\tau(1-d)}$ [9], thus the number of bits that we need in order to compute $s$ is no more than $1389 + 30744\,\tau_\Delta$.

In order to compare two radii $s_1$ and $s_2$, which are roots of polynomials $R_1$ and $R_2$ respectively, we need a bound for $|s_1 - s_2|$. Since $|s_1 - s_2| \geq sep(R_1R_2)$, where the polynomial $R_1R_2$ has degree 368 and coefficient bit size $8 + 336\tau_\Delta$, it follows that the number of bits needed is $1508 + 30324\tau_\Delta$. This is tight, because the system has optimal mixed volume [3].

In computing the implicit representation the bit size increases by a factor of 6. If the parametric input coefficients have $\tau$ bits, then $\tau_\Delta = 6\tau$. If the order of convergence of our method is $\phi$, then the number of iterations needed is $\log_\phi(1508 + 181944\tau)$.

## 4 Implementation and experiments

A reference implementation with parametric ellipses has been done in Maple 9. We have implemented a small algebraic number package that performs exact univariate real root isolation, comparison and sign evaluation of univariate (bivariate) expressions over one (two) algebraic number(s), using Sturm sequences and interval arithmetic over $\mathbb{Q}$.

We speed up the subdivision by noticing that $\mathcal{S}_{trs}$ is strictly monotone in the starting interval $[a, b]$ and has a unique simple real root in it. So we use Brent's method with theoretical convergence rate $\phi = 1.618$ [1]. Let $[a, b]$ be any interval and $m = \frac{a+b}{2}$. The new endpoint is $x = m + \frac{P}{Q}$, where $R = \frac{\mathcal{S}_{trs}(m)}{\mathcal{S}_{trs}(b)}$, $S = \frac{\mathcal{S}_{trs}(m)}{\mathcal{S}_{trs}(a)}$, $T = \frac{\mathcal{S}_{trs}(a)}{\mathcal{S}_{trs}(b)}$, $P = S(T(R - T)(b - m) - (1 - R)(m - a))$ and $Q = (T - 1)(R - 1)(S - 1)$. If $x \notin [a, b]$ then the new estimation is $m$. However, in practice we observe a fast growth of bitsize, if we use this method with an exact number type, i.e. rationals.

Based on this reference implementation in Maple, we implemented the algorithm in C++ using CORE. While still in a preliminary stage, it is faster than the Maple implementation. In this case it is possible to certify our algorithm based on constructive root separation bounds e.g. [2, 10], which should be tighter than the static bounds now used. In this implementation, the input quantities are of type BigRat (rational numbers), while the endpoints of the interval that expresses the Voronoi circle are BigFloats (multiprecision floating point). Evaluation of $Q_1 \pm Q_2$ in (1) is performed using CORE::Expr constructs after converting $t$ from BigFloat to BigRat. This, along with $B(t, r)$ seem to be the most heavy computations, due to growth of bitsize. In a future improvement we will use guaranteed precision arithmetic with BigFloats. Another improvement will be to use information from previous iterations (where $t$ has fewer correct bits) in order to solve $B(t, r)$ and determine the sign of $Q_1$ and $Q_2$.

We performed several preliminary experiments with different triplets of ellipses and circles. We consider a query ellipse (circle) with its centre moving along a line and measure the time taken to decide its relative position wrt the Voronoi circle. Among the various configurations, there were both degenerate and non-degenerate cases.

We did our experiments on a P4 2.6GHz. In fig. 4 we present the times for 2 test suites, where the ellipses have 10-bit coefficients in their parametric form. The first graph involves ellipses that do not share a common Voronoi circle with the query one, whose center moves along the line $y = -x$ (fig. 5 left). Notice that the time increases as we approach a degenerate configuration. Although the hardest cases took about 5s, in 90% of the cases we can decide in less than 2.5s. The C++ implementation without any compiler optimizations is 2 times faster than Maple. The second graph involves circles (fig. 5 right). The peak corresponds to nearly degenerate configurations, running in 38s and 200 iterations. In all other cases the timings are less than 2.5s. Again, the C++ implementation is 3-4 times faster.

Our implementations are exact but can also run with any prescribed precision, e.g. for rendering purposes. In particular, a much faster execution is possible for the above algorithms if we restrict ourselves to machine precision, as in [5].

Solving the algebraic system $B(t, r) = B(t, s) = B(r, s) = 0$ with the SYNAPS [8] package of multivariate subdivision in 3 msec to 1 min, depending on how large the initial domain was. Moreover, we used PHCpack, to solve the system of $\Delta_i$'s, in about 36 seconds. These running times indicate that our dedicated solver sometimes outperforms generic solvers on the algebraic system. Moreover, solving the algebraic system alone does not suffice to completely decide the predicate, contrary to our algorithm.

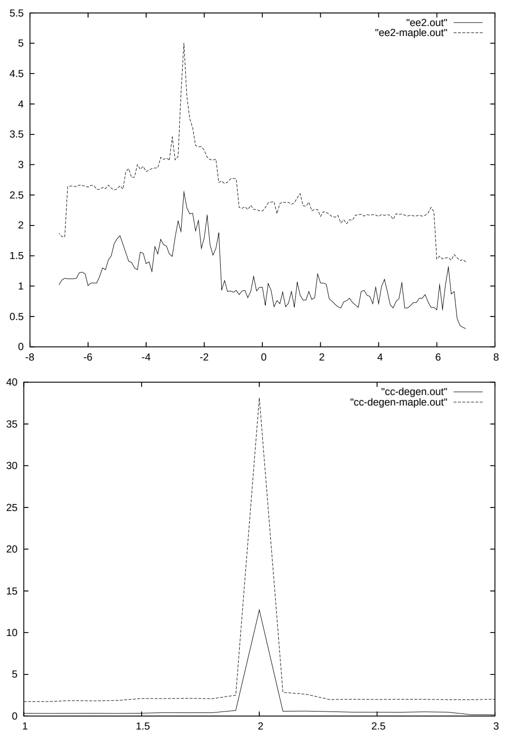


Figure 4: Execution time as function of the position of the query ellipse's (circle's) center. The solid line corresponds to C++, the dotted one to Maple.

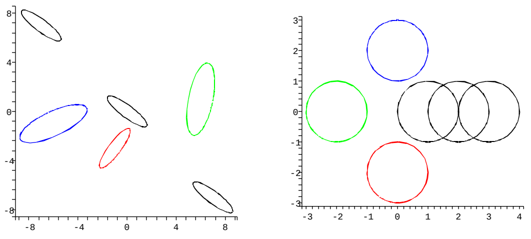


Figure 5: Test suites

## 5 Future work

Our final goal is a CGAL implementation of the Voronoi diagram of ellipses. Working in C++ may allow us to use one of the powerful interval arithmetic packages. We tried the iCOs interval-arithmetic solver [2] on the system of $\Delta_i$'s with bitsize 60. It detects a degeneracy in about 213 sec on a 1GHz P3.

The most difficult part of the implementations is the detection of a degeneracy. Although near-degenerate inputs can be handled quite efficiently, real degeneracies exploit the separation bound and need a large number of iterations. Then, the computed quantities grow too large. We are currently trying to optimize the inner loop. In this direction, we are looking for better ways to perform certified sign computation of $Q_1 \pm Q_2$, as well as better separation bounds using geometric arguments and exploiting the algebraic approach.

**Acknowledgments** All authors acknowledge partial support by IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-006413-2 (ACS - Algorithms for Complex Shapes) and by PYTHAGORAS, project 70/3/7392 under the EPEAEK program funded by the Greek Ministry of Educational Affairs and EU. GT is partially supported by State Scholarship Foundation of Greece, Grant No. 4631.

## References

[1] R. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall, Englewood Cliffs, N.J., 1973.

[2] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A Separation Bound for Real Algebraic Expressions. In *ESA*, volume 2161 of *LNCS*, pages 254–265. Springer, 2001.

[3] I. Emiris and G. Tzoumas. Algebraic study of the Apollonius circle of three ellipses. In *Proc. Europ. Works. Comp. Geom.*, pages 147–150, Holland, 2005. Also: Poster session, CASC'05, Greece. To appear in *SIGSAM Bulletin*.

[4] I. Emiris, G. Tzoumas, and E. Tsigaridas. The predicates of the Voronoi diagram of ellipses. *Symp. of Comp. Geom.*, 2006. To appear. Available from http://www.di.uoa.gr/∼geotz/.

[5] 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, Cambridge, Massachusetts, 2005. Best paper award.

[6] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A CORE library for robust numeric and geometric computation. In *15th ACM Symp. on Computational Geometry*, 1999.

[7] M. Karavelas and M. Yvinec. Voronoi diagram of convex objects in the plane. In *Proc. ESA*, pages 337–348, 2003.

[8] B. Mourrain, J. P. Pavone, P. Trébuchet, and E. Tsigaridas. SYNAPS, a library for symbolic-numeric computation. In *8th Int. Symposium on Effective Methods in Algebraic Geometry, MEGA*, Sardinia, Italy, May 2005. to appear.

[9] C. Yap. *Fundamental Problems of Algorithmic Algebra.* Oxford University Press, New York, 2000.

[10] C. Yap. On guaranteed accuracy computation. In F. Chen and D. Wang, editors, *Geometric Computation*, volume 11 of *Lect. Notes Series Comp.* World Scientific, 2004.

[2]http://www-sop.inria.fr/coprin/ylebbah/icos/

# Voronoi diagrams in Cgal

Menelaos I. Karavelas[*]

**Abstract**

In this paper we describe a generic `C++` adaptor[1], that adapts a 2-dimensional triangulated Delaunay graph and to the corresponding a Voronoi diagram, represented as a doubly connected edge list (DCEL) data structure. Our adaptor has the ability to automatically eliminate, in a consistent manner, degenerate features of the Voronoi diagram, that are artifacts of the requirement that Delaunay graphs should be triangulated even in degenerate configurations. Depending on the type of operations that the underlying Delaunay graph supports, our adaptor allows for the incremental or dynamic construction of Voronoi diagrams and can support point location queries. Our code will appear in the next public release of Cgal.

## 1 Introduction

A Voronoi diagram on the plane is typically defined for a set of planar objects, also called sites in the sequel, and a distance function that measures the distance of a point $x$ in $\mathbb{R}^2$ from an object in the object set. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be our set of sites and let $\delta(x, S_i)$ denote the distance of a point $x \in \mathbb{R}^2$ from the site $S_i$. Given two sites $S_i$ and $S_j$, the set $V_{ij}$ of points that are closer to $S_i$ than to $S_j$ with respect to the distance function $\delta(x, \cdot)$ is simply the set: $V_{ij} = \{x \in \mathbb{R}^2 : \delta(x, S_i) < \delta(x, S_j)\}$. We can then define the set $V_i$ of points on the plane that are closer to $S_i$ than to any other object in $\mathcal{S}$ as $V_i = \bigcap_{i \neq j} V_{ij}$. The set $V_i$ is said to be the *Voronoi cell* or *Voronoi face* of the site $S_i$. The locus of points on the plane that are equidistant from exactly two sites $S_i$ and $S_j$ is called a *Voronoi bisector*. A point that is equidistant to three or more objects in $\mathcal{S}$ is called a *Voronoi vertex*. A simply connected subset of a Voronoi bisector is called a *Voronoi edge*. The collection of Voronoi faces, edges and vertices is called the *Voronoi diagram* of the set $\mathcal{S}$ with respect to the distance function $\delta(x, \cdot)$, and it is a subdivision of the plane.

We typically think of faces as 2-dimensional objects, edges as 1-dimensional objects and vertices as 0-dimensional objects. However, this may not be the case for several combinations of sites and distance functions (for example points in $\mathbb{R}^2$ under the $L_1$ or the $L_\infty$ distance can produce 2-dimensional Voronoi edges). Moreover, the cell of a site can in general consist of several disconnected components (e.g., in the multiplicatively weighted Euclidean Voronoi diagram). In this paper we are going to restrict ourselves to Voronoi diagrams that have the property that the Voronoi cell of each site is a simply connected region of the plane. We are going to call such Voronoi diagrams *simple Voronoi diagrams*. Examples of simple Voronoi diagrams include the usual Euclidean Voronoi diagram of points, the Euclidean Voronoi diagram of a set of disks on the plane, the Euclidean Voronoi diagram of a set of disjoint convex objects on the plane, or the power (Laguerre) diagram for a set of circles on the plane. In fact every instance of an *abstract Voronoi diagram* in the sense of Klein [2] is a simple Voronoi diagram in our setting. In the sequel when we refer to Voronoi diagrams we refer to simple Voronoi diagrams.

## 2 Adapting triangulated Delaunay graphs

In many applications we are not really interested in computing the Voronoi diagram itself, but rather its dual graph, called the *Delaunay graph*. In general the Delaunay graph is a planar graph, each face of which consists of at least three edges. Under the non-degeneracy assumption that no point in the plane is equidistant to more than three sites, the Delaunay graph is a planar graph with triangular faces. In certain cases this graph can actually be embedded with straight line segments in which case we talk about a triangulation (e.g., the Euclidean Voronoi diagram/Delaunay triangulation of points, or the power diagram/regular triangulation of a set of circles). Graphs of non-constant non-uniform face complexity can be undesirable in many applications, so we typically end up triangulating the non-triangular faces of the Delaunay graph.

Choosing between computing the Voronoi diagram or the (triangulated) Delaunay graph is a major decision while implementing an algorithm. It heavily affects the design and choice of the different data structures involved. Although in theory the two approaches are entirely equivalent, it is not so straightforward to go from one representation to the other.

[*]Department of Applied Mathematics, University of Crete; `mkaravel@tem.uoc.gr` and Institute of Applied and Computational Mathematics, Foundation for Research and Technology - Hellas.

[1]An adaptor is a class or a function that transforms one interface into a different one.

The objective for our adaptor is to provide a generic way of going from triangulated Delaunay graphs to planar subdivisions represented through a DCEL data structure. Although the look and feel is that of a DCEL data structure, internally we keep the graph data structure representing triangular graphs.

The adaptation might seem straightforward at a first glance, and this is true if our data do not contain degenerate configurations. The situation becomes complicated whenever we want to treat the artifacts introduced in our representation due to these degenerate configurations. Suppose for example that we have a set of sites that contains subsets of sites in degenerate positions. The dual of the computed triangulated Delaunay graph is a Voronoi diagram that has all its vertices of degree 3, and for that purpose we are going to call it a *degree-3 Voronoi diagram* in order to distinguish it from the true Voronoi diagram of the input sites. A degree-3 Voronoi diagram can have degenerate features, namely Voronoi edges of zero length, and/or Voronoi faces of zero area, and do not correspond to the true geometry of the Voronoi diagram.

The manner that we treat such issues is by defining an *adaptation policy*. The adaptation policy is responsible for determining which features in the degree-3 Voronoi diagram are to be rejected and which not. The policy to be used can vary depending on the application or the intended usage of the resulting Voronoi diagram. What we care about is that firstly the policy itself is consistent and, secondly, that the adaptation is also done in a consistent manner. The latter is the responsibility of the adaptor we provide, whereas the former is the responsibility of the implementor of a policy. We currently provide two types of adaptation policies, which are discussed in Section 5.

Delaunay graphs can be mutable or non-mutable. By mutable we mean that sites can be inserted or removed at any time, in an entirely on-line fashion. By non-mutable we mean that once the Delaunay graph has been created, no changes, with respect to the set of sites defining it, are allowed. If the Delaunay graph is a non-mutable one, then the Voronoi diagram adaptor is a non-mutable adaptor as well. If the Delaunay graph is mutable then the question of whether the Voronoi diagram adaptor is also mutable is slightly more complex to answer. In Section 6 we discuss the issue in detail.

## 3 Software design

The class `Voronoi_diagram_2<DG,AT,AP>` implements our generic adaptor. It is parametrized by three template parameters which are required to be models of corresponding concepts (see Fig. 1). The first template parameter must be a model of the `DelaunayGraph_2` concept, which corresponds to the interface required from a class representing a Delaunay graph. Currently, all classes of CGAL that represent Delaunay graphs are models of this concept, namely, Delaunay triangulations, regular triangulations, Apollonius graphs and segment Delaunay graphs [1]. The second template parameter must be a model of the `AdaptationTraits_2` concept, which is responsible for accessing the geometric information needed from the specific Delaunay graph in order to perform the adaptation. We discuss this concept in detail in Section 4. Finally, the third template parameter must be model of the `AdaptationPolicy_2` concept, which refers to the policy used to perform the adaptation. This concept is discussed in detail in Section 5.

The `Voronoi_diagram_2<DG,AT,AP>` class has been intentionally designed to provide an interface similar to CGAL's arrangements: Voronoi diagrams are special cases of arrangements after all. The interfaces of the two classes, however, could not be identical. The reason is that arrangements in CGAL do not yet support more than one unbounded faces, or equivalently, cannot handle unbounded curves. On the contrary, a Voronoi diagram defined over at least two generating sites, has at least two unbounded faces.

On a more technical level, the `Voronoi_diagram_2<DG,AT,AP>` class imitates the representation of the Voronoi diagram (seen as a planar subdivision) by a DCEL (Doubly Connected Edge List) data structure. We have vertices (the Voronoi vertices), halfedges (oriented versions of the Voronoi edges) and faces (the Voronoi cells). We can perform all standard operations of the DCEL data structure: go from a halfedge to its next and previous in the face; go from one face to an adjacent one through a halfedge and its twin (opposite) halfedge; walk around the boundary of a face; enumerate/traverse the halfedges incident to a vertex from a halfedge, access the adjacent face; from a face, access an adjacent halfedge; from a halfedge, access its source and target vertices; from a vertex, access an incident halfedge.

In addition to the above possibilities for traversal, we can also traverse the following features through iterators: the vertices of the Voronoi diagram; the edges or halfedges of the Voronoi diagram; the faces of the Voronoi diagram; the bounded/unbounded faces of the Voronoi diagram; the bounded/unbounded halfedges of the Voronoi diagram; the sites defining the Voronoi diagram.

Finally, depending on the adaptation traits passed to the Voronoi diagram adaptor, we can perform point location queries, namely given a point $p$ we can determine the feature of the Voronoi diagram (vertex, edge, face) on which $p$ lies.

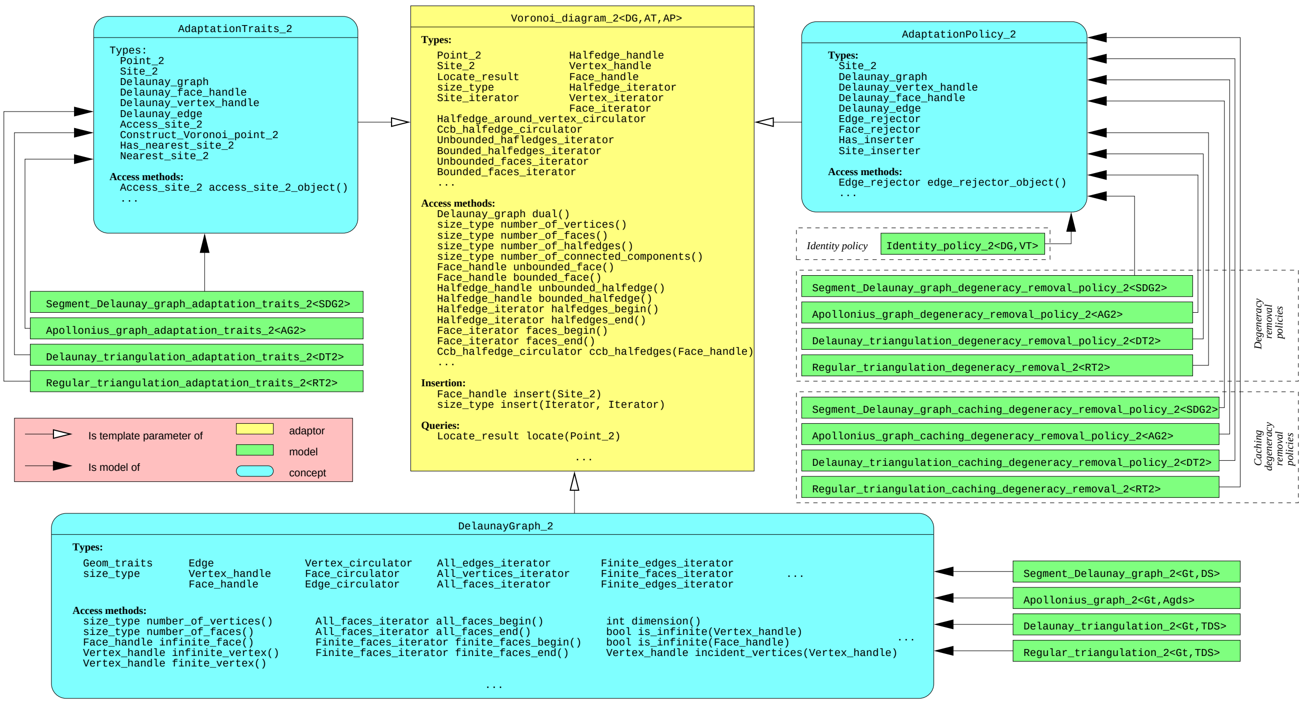

AdaptationTraits_2
Types:
Point_2
Site_2
Delaunay_graph
Delaunay_face_handle
Delaunay_vertex_handle
Delaunay_edge
Access_site_2
Construct_Voronoi_point_2
Has_nearest_site_2
Nearest_site_2
Access methods:
Access_site_2 access_site_2_object()
...
Segment_Delaunay_graph_adaptation_traits_2<SDG2>
Apollonius_graph_adaptation_traits_2<AG2>
Delaunay_triangulation_adaptation_traits_2<DT2>
Regular_triangulation_adaptation_traits_2<RT2>
Is template parameter of
Is model of
adaptor
model
concept
Voronoi_diagram_2<DG,AT,AP>
Types:
Point_2
Site_2
Locate_result
size_type
Site_iterator
Halfedge_handle
Vertex_handle
Face_handle
Halfedge_iterator
Vertex_iterator
Face_iterator
Halfedge_around_vertex_circulator
Ccb_halfedge_circulator
Unbounded_hafledges_iterator
Bounded_halfedges_iterator
Unbounded_faces_iterator
Bounded_faces_iterator
...
Access methods:
Delaunay_graph dual()
size_type number_of_vertices()
size_type number_of_faces()
size_type number_of_halfedges()
size_type number_of_connected_components()
Face_handle unbounded_face()
Face_handle bounded_face()
Halfedge_handle unbounded_halfedge()
Halfedge_handle bounded_halfedge()
Halfedge_iterator halfedges_begin()
Halfedge_iterator halfedges_end()
Face_iterator faces_begin()
Face_iterator faces_end()
Ccb_halfedge_circulator ccb_halfedges(Face_handle)
...
Insertion:
Face_handle insert(Site_2)
size_type insert(Iterator, Iterator)
Queries:
Locate_result locate(Point_2)
...
AdaptationPolicy_2
Types:
Site_2
Delaunay_graph
Delaunay_vertex_handle
Delaunay_face_handle
Delaunay_edge
Edge_rejector
Face_rejector
Has_inserter
Site_inserter
Access methods:
Edge_rejector edge_rejector_object()
...
Identity policy
Identity_policy_2<DG,VT>
Segment_Delaunay_graph_degeneracy_removal_policy_2<SDG2>
Apollonius_graph_degeneracy_removal_policy_2<AG2>
Delaunay_triangulation_degeneracy_removal_policy_2<DT2>
Regular_triangulation_degeneracy_removal_2<RT2>
Degeneracy removal policies
Segment_Delaunay_graph_caching_degeneracy_removal_policy_2<SDG2>
Apollonius_graph_caching_degeneracy_removal_policy_2<AG2>
Delaunay_triangulation_caching_degeneracy_removal_policy_2<DT2>
Regular_triangulation_caching_degeneracy_removal_2<RT2>
Caching degeneracy removal policies
DelaunayGraph_2
Types:
Geom_traits
size_type
Edge
Vertex_handle
Face_handle
Vertex_circulator
Face_circulator
Edge_circulator
All_edges_iterator
All_vertices_iterator
All_faces_iterator
Finite_edges_iterator
Finite_faces_iterator
Finite_edges_iterator
...
Access methods:
size_type number_of_vertices()
size_type number_of_faces()
Face_handle infinite_face()
Vertex_handle infinite_vertex()
Vertex_handle finite_vertex()
All_faces_iterator all_faces_begin()
All_faces_iterator all_faces_end()
Finite_faces_iterator finite_faces_begin()
Finite_faces_iterator finite_faces_end()
int dimension()
bool is_infinite(Vertex_handle)
bool is_infinite(Face_handle)
Vertex_handle incident_vertices(Vertex_handle)
...
...
Segment_Delaunay_graph_2<Gt,DS>
Apollonius_graph_2<Gt,Agds>
Delaunay_triangulation_2<Gt,TDS>
Regular_triangulation_2<Gt,TDS>

Figure 1: The design of the Voronoi diagram adaptor, and the relations between the various concepts, their models and the adaptor.

## 4 The adaptation traits

The `AdaptationTraits_2` concept defines the types and functors required by our adaptor in order to access geometric information from the Delaunay graph. In particular, it defines the type of the generating sites, and provides functors for accessing these sites in the Delaunay graph as well as constructing Voronoi vertices given their dual triangular faces in the Delaunay graph.

Finally, it defines a tag that indicates whether nearest site queries are to be supported by the Voronoi diagram adaptor. If such queries are to be supported, a corresponding functor is also required. Given a query point, the nearest site functor should return information related to how many and which sites of the Voronoi diagram are at equal and minimal distance from the query point. This way of abstracting the point location mechanism allows for multiple different point location strategies, which are passed to the Voronoi diagram adaptor through different models of the `AdaptationTraits_2` concept. The point location queries of the `Voronoi_diagram_2<DG,AT,AP>` class uses internally this nearest site query functor.

Along with our adaptor we provide four adaptation traits classes, all of which support nearest site queries. These four classes serve as adaptation traits for CGAL's Apollonius graphs, Delaunay and regular triangulations and segment Delaunay graphs, respectively.

## 5 The adaptation policy

When we perform the adaptation of a triangulated Delaunay graph to a Voronoi diagram, a question that arises is whether we want to eliminate certain features of the Delaunay graph when we construct its Voronoi diagram representation. We resolve such issues, in a generic way, via the introduction of an adaptation policy. The adaptation policy is responsible for determining which features in the degree-3 Voronoi diagram are to be rejected and which not. The policy to be used can vary depending on the application or the intended usage of the resulting Voronoi diagram.

The concept `AdaptationPolicy_2` defines the requirements on the predicate functors that determine whether a feature of the triangulated Delaunay graph should be rejected or not. More specifically it defines an `Edge_rejector` and a `Face_rejector` functor that answer the question: “should this edge (face) of the Voronoi diagram be rejected?”.

We have implemented two types of policies that provide two different ways for answering the question of which features of the Voronoi diagram to keep and which to discard. The first one is called the *identity policy* and corresponds to the `Identity_policy_2<DG,VT>` class. This policy is in some sense the simplest possible one, since it does not reject any feature of the Delaunay graph. The Voronoi diagram provided by the adaptor is the true dual (from the graph-theoretical point of view) of the triangulated Delaunay graph adapted.

The second type of policy we provide is called *degeneracy removal policy*. If the set of sites defining the triangulated Delaunay graph contains subsets of sites in degenerate configurations, the graph-theoretical dual of the triangulated Delaunay graph has edges and potentially faces that are geometrically degenerate. By that we mean that the dual of the triangulated Delaunay graph can have Voronoi edges of zero length or Voronoi faces/cells of zero area. Such features may not be desirable, in which case we would like to eliminate them. The degeneracy removal policies eliminate exactly these features. Along with our Voronoi diagram adaptor we provide four degeneracy removal policies, namely for Apollonius graphs, Delaunay triangulations, regular triangulations and segment Delaunay graphs.

A variation of the degeneracy removal policies are the *caching degeneracy removal policies*. In these policies we cache the results of the edge and face rejectors. Such policies really pay off when we have a lot of degenerate data in our input set of sites. This is due to the fact that detecting whether a Voronoi edge or a Voronoi face is degenerate implies computing the outcome of a predicate in a possibly degenerate or near degenerate configuration, which is typically very costly (compared to computing the same predicate in a generic configuration). We provide four caching degeneracy removal policies, one per degeneracy removal policy mentioned above.

## 6 Mutable vs. non-mutable policies

In addition to the edge and face rejectors the adaptation policy defines a boolean tag, the `Has_inserter` tag. Semantically, this tag determines if the adaptor is allowed to insert sites in an on-line fashion (on-line removals are not yet supported). In the former case, i.e., when on-line site insertions are allowed, an additional functor is required, the `Site_inserter` functor. This functor takes as arguments a reference to a Delaunay graph and a site, and inserts the site in the Delaunay graph. Upon successful insertion, a handle to the vertex representing the site in the Delaunay graph is returned. In our discussion of the adaptation policies, we did not indicate the value of the `Has_inserter` tag for the degeneracy removal and caching degeneracy removal policies. The issue is discussed in detail in the sequel.

In Section 2 we raised the question whether the adaptor is a mutable or non-mutable one, in the sense of whether we can add/remove sites in an on-line fashion. The answer to this question depends on: (1) whether the Delaunay graph adapted allows for on-line insertions/removals and (2) whether the associated adaptation policy maintains a state and whether this state is easily maintainable when we want to allow for on-line modifications.

As we mentioned above, the way we indicate if we allow on-line insertions of sites is via the `Has_inserter` tag. A *true* value indicates that our adaptation policy allows for on-line insertions, whereas a *false* value indicates the opposite. Note that these values *do not* indicate if the Delaunay graph supports on-line insertions, but rather whether the Voronoi diagram adaptor should be able to perform on-line insertions or not. This delicate point will be become clearer below.

If the Delaunay graph is non-mutable, the Voronoi diagram adaptor cannot perform on-line insertions of sites anyway. In this case not only degeneracy removal policies, but rather every single adaptation policy for adapting the Delaunay graph in question should have the `Has_inserter` tag set to *true*.

If the Delaunay graph is mutable we can choose between two types of adaptation policies, those that allow these on-line insertions and those that do not. At a first glance it may seem excessive to restrict existing functionality. There are situations, however, where such a choice is necessary. Consider a caching degeneracy removal policy. If we do not allow for on-line insertions then the cached quantities are always valid since the Voronoi diagram never changes. If we allow for on-line insertions the Voronoi diagram can change, which implies that the results of the edge and face degeneracy testers that we have been cached are no longer valid or relevant. In these cases, we need to update the cached values, and ideally we would like to do this in an efficient manner.

For our caching degeneracy removal policies, our choice was made on the grounds of whether we can update the cached results efficiently when insertions are performed. For CGAL's Apollonius graphs, Delaunay triangulation and regular triangulations it is possible to ask what are the edges and faces of the Delaunay graph that are to be destroyed when a query site is inserted. This is done via the `get_conflicts` method provided by these classes. Using the outcome of the `get_conflicts` method the site inserter can first update the cached results (i.e., indicate which are invalidated) and then perform the actual insertion. Such a method does not yet exist for segment Delaunay graphs. We have thus chosen to support on-line insertions for all non-caching degeneracy removal policies, i.e., the caching degeneracy removal policy for segment Delaunay graphs does not support on-line insertions, whereas the remaining three caching degeneracy removal policies support on-line insertions.

## References

[1] The CGAL homepage. `http://www.cgal.org/`.

[2] R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1989.

## Author Index